Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação
DCC059 - Teoria dos Grafos Semestre 2016-3

Abordagens Heurísticas para o Problema do Caixeiro Viajante Preto e Branco

Gabriel Dias de Abreu

Letícia Florentino Pires

Professor: Stênio Sã Rosário F. Soares

Relatório do trabalho final de Teroria dos Grafos, parte integrante da avaliação da disciplina.

Juiz de Fora

Dezembro de 2016

1 Introdução

Problema do Caixeiro Viajante Preto e Branco (PCV-PB) [5] é definido em um grafo particionado em conjuntos disjuntos de vértices pretos e brancos, e consiste em encontrar um ciclo hamiltoniano de custo mínimo obedecendo as restrições de cardinalidade e de comprimento. A primeira se refere a quantidade de vértices brancos que podem existir entre dois vértices pretos consecutivos, a segunda refere-se a distância permitida entre esses dois citados vértices pretos.[6] O PCV BP é NP-Dificil,sendo abordado por meio de heurísticas. [6] É aplicável em problemas de roteamento e escalonamento tais como escalonamento de aeronaves e tripulações aéreas, configurações de redes de telecomunicações. [6]

2 Metodologia utilizada

2.1 Estruturas de dados utilizadas

- A estrutura de dados usada como base para o armazenamento do conjunto de vértices e arestas consiste em um vetor e várias listas. O vetor é composto por estruturas do tipo "Vértice" que têm ID e peso, dentre outros métodos há o "incluirAresta" que forma uma lista encadeada de arestas que têm esse vértice como um dos seus endpoints. Uma observação importante é que o "ID" age tanto como indíce do vetor quanto informação do vértice. Quando o vértice não existe, ele tem valor -1.

  As listas de arestas são compostas por estruturas do tipo "Aresta" com os campos ID e peso, que são vértices da vizinhança de um dado elemento do vetor.

  Essa estrutura foi escolhida pois a busca por vértices no vetor é O(1).

- No decorrer da implementação foram usadas estruturas lineares básicas da STL do C++: vector e list. Também foram usadas queues e stacks. O uso das mesmas ocorreu devido as suas propriedades intrínsecas, além da praticidade e estabilidade oferecidas.

2.2 Abordagens algoritmicas usadas na solução

Sendo o PCV-BP um problema NP-Díficil, usamos a metaheurística GRASP (Greedy Randomized Adaptive Search Procedure) proposta em 1995 por Thomas Feo e Maurício Resende [2], embora sua origem remeta a trabalhos anteriores, como Lin e Kerninghan [4], Hart e Shogan [3] e ainda Feo e Resende [2]. Esse método é composto de duas fases :

- **Construtiva: constrói uma solução viável para o problema de forma gulosa-aleatória.**
- **Busca local: procura melhorar a qualidade da solução gerada anteriormente.**

2.3 Heurísticas propostas nesse trabalho

O Algoritmo 1 é responsável pela primeira parte da construção da solução, gerando um ciclo hamiltoniano de custo minímo constituido apenas por vértices pretos.

Algorithm 1 Gera Hp

```
1: function GERA HP (α)                ▷ α - fator limitante das escolhas de candidatos
2:     Seja Visitados[1...maxVertices]
3:     for i = 1 to maxVertices do
4:         Visitados[i] ← false
5:     end for
6:     Cria lista de soluções : s
7:     Cria vetor de pretos adjacentes : g
8:     restricao ← α *numerodepretos
9:     Visitados[v] ← true
10:    s ← melhor vértice
11:    for p = i to numerodePretos − 1 do
12:        Cria lista adj com os adjacentes de v
13:        for todo elemento de adj do
14:            if vértice é preto & não visitado then
15:                g ← v
16:            end if
17:        end for
18:        Ordena não crescentemente g por critério de pesos
19:        Escolhe um k em g
20:        s ← k
21:        Visitados[k] ← true
22:    end for
23:    retorna s
24: end function
```

O Algoritmo 2 insere os vértice brancos no ciclo já formado pelo Algoritmo 1, obdecendo a restrição de cardinalidade e de distância.

Algorithm 2 Insere Brancos

```
function INSERE BRANCOS(solucao, α ) ▷ solucao - lista formada pelo algoritmo 1,
α - fator limitante das escolhas de candidatos
    Cria vetor qtdBrancoEntrePreto
    Cria vetor b
    restricao ← α ∗tamanhodesoluao
    for para todo v , pertencente ao conjunto de vértices brancos do
        Cria lista adj com os adjacentes de v
        for todo elemento v de adj do
            if vértice é preto & obedece a cardinalidade & obedece a distância then
                b ← v
            end if
        end for
        Ordena não crescentemente b por critério de pesos
        Escolhe um k em b
        for todo elemento de s do
            if s = k then
                s ← v
                qtdBrancoEntrePreto[k]++
            end if
        end for
    end for
    retorna s
end function
```

Algorithm 3 Algoritmo Guloso

```
function GULOSO ▷
    solucaoParcial ← GeraHP (0)
    solucao ← InsereBranco(solucaoParcial,0)
    Retorna solucao
end function
```

Algorithm 4 Algoritmo Guloso Randomizado

```
function GULOSO RANDOMIZADO(α , numeroIteracoes)    ▷ α - fator limitante das escolhas de candidatos, numeroIteracoes - número de Iterações
    melhorSolucao ← geraHp(α)
    InsereBranco(melhorSolucao, α)
4:  melhorCusto ← calculaCusto(melhorSolucao)
    while numeroIteracoes do
        solucao ← geraHp(α)
        InsereBranco(solucao, α)
     custo= calculaCusto(solucao)
8:      if custo < melhorCusto then
            melhorSolucao ← solucao
            melhorCusto ← custo
        end if
12: end while
     retorno melhorSolucao
end function
```

Os algoritmos 1 e 2 podem ser gulosos ou gulosos randomizados, dependendo do argumento α passado para os mesmos : se $\alpha = 1$ é puramente guloso , $\alpha = 0$ puramente aleatório. A função encapsuladora do Algoritmo 3, encarrega de mante-los puramente gulosos por meio do parâmetro $\alpha = 0$. Caso seja guloso randomizado, o Algoritmo 4 tem $\alpha = 0{,}25$ como parâmetro, e itera-se por vezes suficientes até que se consiga uma solução com bom custo. É perceptível que em todo o processo os algoritmos 1 e 2 são os núcleos. A seguir, ambos serão explicados.

O intuito dos algoritmos 1 e 2 é formar um ciclo hamiltoniano de custo mínimo. Para isso, primeiro forma-se um ciclo hamiltoniano de custo mínimo de pretos escolhendo sempre o melhor elemento para a solução, sendo o Algoritmo 1 responsável por essa parte. Para isso cria-se uma lista **s**, onde ficarão os vértices pretos que compõem as melhores arestas (linha 6), uma lista auxiliar **adj** de vértices que compõem a vizinhança do vértice selecionado (linha 12), e um vetor de vértices pretos **g** preenchidos a partir do refinamento da lista **adj**. A prior escolhemos o melhor vértice (linha 10) a partir do qual toda iteração se desenvolve, após esse passo os vértices serão escolhidos a partir da escolha de um vértice da seleção dos melhores elementos (ou de todo o conjunto, a depender de α) (linhas 8 e 19) do vetor **g** ordenado (linha 18). A ordenação é feita por uma função do próprio C++,sendo O(nlogn). O algortimo retorna a lista **s**.

No Algoritmo 2, completamos a solução com os vértices brancos. Nele, itera-se por todos os vértices brancos (linha 4) e forma-se uma lista com os seus adjacentes **adj** , seleciona-se

os melhores vértices pretos dessa vizinhança no vetor **b** respeitando a cardinalidade e a distância entre pretos (linhas 7 - 11). Itera-se sobre **b** e encontra-se os vértices pretos comuns com a solução gerada pelo Algoritmo 1, o vértice branco vizinho desse vértice entra na solução final (linha 16) . Retorna-se a solução final (linha 13).

O Algoritmo 5 contém a junção de todos os explicados acima, acrescentando a técnica de adaptativa de escolha do melhor α do vetor de α sugerido, através de um vetor associado de probabilidades (linha 2). O elemento cuja aleatoriedade é a escolhida é obtido a partir do decremento dessa probabilidade até que encontra-se um elemento do vetor **nSi** que seja igual (linhas 9 - 11). Então gera-se uma nova solução a partir do α ao qual se refere (linhas 11- 14) e verifica se a mesma é melhor (linha 15). Retornando a melhor solução (linha 20)

Algorithm 5 Algoritmo Guloso Randomizado Reativo

function ALGORITMO GULOSO RANDOMIZADO REATIVO(maxIteracoes) ▷
maxIteracoes - número máximo de iterações
Cria um vetor **si** cujos elementos são α
Cria um vetor **nSi** cujos elementos são as probanilidade dos respectivos **a** acontecerem
numeroTotalSucesso ← 10
5: i = número aleatório de 1 até 10
melhorSolucao ← geraHp(**nSi**[i])
melhorCusto ← calculaCusto(melhorSolucao)
while numeroIteracoes **do**
for um **o** aleatório no range de numeroTotalSucessos até 0 **do**
10: decrementa-se **o** até que seja igual a um elemento em **nSi**
end for
solucao ← geraHp(**si**[i])
insereBrancos(solucao,**si**[i])
custo ← calculaCusto(solucao)
15: **if** custo < melhorCusto **then**
melhorSolucao ← solucao
melhorCusto ← custo
end if
end while
20: retorno melhorSolucao
end function

3 Experimentos computacionais

3.1 Ambiente Computacional

Para a realização dos testes computacionais, todos os algoritmos foram implementados na linguagem C++, compilados com g++ 6.2.1 e executados em uma máquina com processador Intel(R) i5 4210u @ 1,7GHz à 2,7GHz e 4 GB de RAM DDR3, no ambiente Linux na distribuição rolling release Archlinux. Para análise estatística dos dados foi utilizado o software Libre Office Calc(R).

3.2 Métrica para Avaliação dos Algoritmos

A métrica sugerida para avaliar o desempenho [7]. A medida esta descrita abaixo onde f_{metodo} é o valor do custo obtido pelo algoritmo que foi implementado nesse trabalho e f_{melhor} é o melhor encontrado de todos métodos no trabalho em questão e na segunda tabelaf_{melhor} é a melhor solução obtida pelo algoritmo da literatura.

3.3 Problemas Teste

As instâncias utilizadas para teste nesse trabalho foram escolhidas a partir de um conjunto montado por [1]. Escolhemos 4 , 3 e 3 instâncias de pequeno, médio e grande porte, respectivamente , todas elas constituem grafos simples ponderados e não direcionados, A escolha se limitou a grafos com no máximo 200 vértices, pois o problema PCV-BP é reconhecidamente NP-Díficil [6]. A descrição de cada instância escolhida está detalhada na tabela abaixo :

Tabela 1: Descrição das instâncias escolhidas.

Instância	|V|	|P|	Cardinalidade	Distância
01	51	10	9	114
02	51	10	14	200
03	51	10	9	100
04	51	15	12	153
11	100	20	10	5168
12	100	20	9	9044
13	100	20	14	4522
16	200	40	9	1479
17	200	40	14	2588
18	200	40	14	1294

Seguem as tabelas com os resultados :

Esta tabela exibe o DPR% em relação ao melhor resultado encontrado aplicando a heurística proposta nesse trabalho.

Tabela 2: Descrição dos identificadores utilizados nas tabelas completas de resultados relacionadas aos construtivos.

Identificador	Descrição
id	Instância
β	Valor da melhor solução
σ_g	Desvio percentual para guloso
σ_{gr}	Desvio percentual para guloso randomizado
σ_{grr}	Desvio percentual para guloso randomizado reativo

id	β	σ_g	σ_{gr}	σ_{grr}
01	773.367	3,19%	19,63%	0%
02	779	2,44%	17,45%	0%
03	774.9	2,98%	19,21%	0%
04	663.6	2,77%	26,21%	0%
11	51595	0%	51,98%	2,06%
12	5253	0%	51,35%	0,94%
13	51595	0%	52,23%	2,66%
16	22790	0%	130,58%	35,97%
17	22790	0%	128,82%	36,38%
18	22790	0%	130,21%	36,76%

Tabela 3: Tabela - Considera-se o desvio em relação melhor valor do nosso trabalho

A tabela abaixo exibe o DPR% em relação ao melhor valor da literatura, que encontra-se na coluna "best"da Tabela D1 página 148 de [1].

id	β	σ_g	σ_{gr}	σ_{grr}
01	435,40	83,28%	112,49%	77,62%
02	428,87	86,07%	113,34%	81,64%
03	436,87	82,66%	111,45%	77,38%
04	428,87	59,02%	95,28%	54,73%
11	22139,10	133,05%	254,19%	137,85%
12	22593,90	132,5%	251,88%	134,68%
13	22139,10	133,05%	254,76%	139,24%
16	1132,80	1911,83%	4538,92%	2635,54%
17	10971,10	107,73%	375,33%	183,3%
18	10969,10	107,77%	378,29%	184,14%

Tabela 4: Tabela - Considera-se o desvio em relação melhor valor da literatura

A tabela abaixo exibe o tempo médio de execução para cada instância em cada abordagem. A unidade base de tempo exibida é em minutos.

Tabela 5: Descrição dos identificadores utilizados nas tabelas completas de resultados relacionadas aos construtivos.

Identificador	Descrição
id	Instância
t_g	Média de tempo para guloso
t_{gr}	Média de tempo para guloso randomizado
t_{grr}	Média de tempo para guloso randomizado reativo

id	$t_g[m]$	$t_{gr}[m]$	$t_{grr}[m]$
01	00:00,000	00:01,565	00:08,493
02	00:00,000	00:01,711	00:09,234
03	00:00,000	00:01,947	00:08,228
04	00:00,000	00:01,960	00:10,730
11	00:00,000	00:13,541	01:09,012
12	00:00,000	00:13,335	00:59,078
13	00:00,000	00:09,749	01:01,850
16	00:02,000	01:35,994	03:53,841
17	00:02,000	00:51,459	03:28,391
18	00:01,000	00:50,083	03:35,621

Tabela 6: Tabela - Média de tempo de execução

4 Conclusões

Confrontando os resultados de nosso trabalho com os da literatura [1], a partir da Heurística aqui proposta e abrindo mão de técnicas de refinamento como busca local e metaheurísticas mais refinadas, chegamos a resultados bem distantes da mesma. A menor distância foi de 54,73% no método Guloso Randomizado Reativo em uma instância pequena (Tabela 4). Essa é uma tendência do nosso algoritmo para as três técnicas utilizadas, em todas as instâncias menores tivemos resultados menos distantes da aprsentada pela best da literatura (Tabela 4).

Acredita-se que a falta de técnicas mais acessíveis de refinamento teria sido a causa maior dessa discrepância, ainda ressaltamos que um número maior de iterações de nossas técnicas talvez pudesse fornecer resultados melhores, já que aqueles obtidos ainda apresentavam uma melhora com o aumento das iterações, mas ainda assim fixamos as iterações em um valor médio como descrito nas instruções. Esse impacto pode ser percebido principalmente

nos resultados sobre o método do Guloso Randomizado (Tabela 4).

Nossa heurística trabalhava ao mesmo tempo com duas otimizações, sendo elas o caixeiro-viajante e a seguir a técnica de inserção, o alfa otimizado a partir de uma, não deve consequentemente atender as duas, o que provavelmente acumulou um erro na função objetivo(custo) em nossa solução, em questões de otimalidade.

É importante ressaltar que as técnicas de construção de solução utilizadas na literatura [6] e [1] são complexas e por isso optamos por uma heurística mais simples e direta, ainda que o resultado encontrado não tenha sido satisfatório. Sabemos que a viabilização do PCV-BP é em si um problema complicado, por isso entendemos que partir de ciclos hamiltanianos formados apenas por vértices pretos aumentariamos a possibilidade de viabilização, mesmo que pelo pesado custo dos caminhos se mostrarem piores, como visto nas avaliações estatísticas (Tabela 4).

Conseguimos observar um problema que poderia levar séculos para ser resolvido sem inteligência computacional, ser resolvido em alguns milisegundos, como mostra a tabela 6, de forma satisfatória, e com algumas técnicas essa solução foi se tornando melhor, sendo que ainda existem diversos métodos que fogem ao escopo desse trabalho que poderiam ter melhorado a mesma.

Referências

[1] Paôla Pinto Cazetta. Abordagens heurísticas para tratar o problema do caixeiro viajante preto e branco. 2015.

[2] M. G.; Feo, T. A. Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, page 109–133, 1995.

[3] J.P. Hart and A.W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters,*, page 107–114, 1987.

[4] B. W. Lin, S. Kernighan. An effective heuristic algorithm for the traveling- salesman problem. *Operations research*, page 498–516, 1973.

[5] Frederic Semeta; Melanie Bourgeois, Gilbert Laporte;. Heuristics for the black andwhite traveling salesman problem. *Computers Operations Research*, 1999.

[6] Maciel A. C. M. Martinhon C. A. Ochi L. S. Heursticas e metaheursticas para o problema do caixeiro viajante branco e preto. *XXXVII Simpsio Brasileiro de Pesquisa Operacional*, 44:17–18, 2005.

[7] R. Minella G. Vallada, E.; Ruiz. Minimising total tardiness in the m- machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers Operations Research*, 44:35(4):1350–1373, 2008.

A Anexo I

Um dos problemas propostos para a elaboração base desse trabalho é a verificação se o Grafo é K-Conexo. Tratando-se um problema da classe NP, abordamos-o de maneira simplificada através do seguinte algoritmo.

Algorithm 6 eKConexo

```
function EKCONEXO                                                    ▷
    if o Grafo é completo then
        é K-Conexo.
        if o Grafo não é completo ∨ número de vértices = 1 then
 5:         não é K-Conexo.
        else
            Ordena a sequência de graus
            cria lista g com vizinhança aberta de um vértice
            while g do
10:             Exclui um vértice
                if Grafo não continua conexo then
                    não é K-Conexo
                end if
            end while
15:         Imprime "não é possível afirmar"
    end if
end if
end function
```