Agendamento de Refeições em Restaurante Universitário

Lucas Dalbonio, Mayla Lomelino
Orientador: Ricardo Cordeiro Correa

[1]Departamento de Tecnologias e Linguagens –
Universidade Federal Rural do Rio de Janeiro (UFRRJ)
R. Governador Roberto Silveira S/N – Nova Iguaçu –
Rio de Janeiro – RJ – Brasil

correa@ufrrj.br, dalbonio2@gmail.com, mayvic.lomelino@gmail.com

Abstract. *Aiming to improve the management of the university restaurant, an idea of booking the time was grasped. To execute it, graphs were used, where the problem is to partition the graph, maximizing or minimizing an objective function of choice. The project goal is to find a good solution, as closest as possible to the best solution, pleasing the groups and that distributes the meals along time in the best way. Three strategies were used, two of them are based in the degree of each node and the third is based at the adaptation of a population at a ambient. All strategies were tested and the result is that the third strategy is greater. Such result can not be compared to the best possible solution, as this one is not available for us, but is noticeable that between the three strategies, this one is highlighted*

Resumo. *Com o intuito de melhorar o gerenciamento do restaurante universitário, pensou-se em um sistema de agendamento de horários. Para a realização do mesmo, fez-se uso de grafos, onde o problema se resume a particionar o grafo maximizando ou minimizando uma função objetivo escolhida. O projeto visa encontrar uma boa solução, mais perto da melhor possivel, agradando os grupos formados e que distruibua de melhor forma a quantidade de refeições por horários disponíveis. Foram usadas três estratégias distintas, duas se baseiam no grau de cada nó e a terceira na adaptação de uma população em um ambiente. Foram realizados testes distintos utilizando todas as estratégias e chegou-se a conclusão que os resultados encontrados pelo terceiro algoritmo foram superiores. Tal resultado não há como ser comparado ao resultado ótimo pois este não se encontra a nossa disponibilidade, mas é notório que dentre os testados essa abordagem se destaca.*

1. Introdução

O restaurante universitário é um recurso valioso para qualquer universidade, dito isso, o gerenciamento do mesmo reflete diretamente nos gastos da instituição. A motivação desse trabalho é a inclusão de um sistema de agendamento, onde um grupo de pessoas agendaria um horário do restaurante. Esse problema pode ser reduzido para o problema de particionamentos em grafos, onde o nó do grafo é uma pessoa, e as arestas indicam se as pessoas tem afinidade. Com o objetivo de achar uma boa solução para o problema do particionamento, foram feitos testes utilizando 3 algoritmos, com diferentes quantidades de grupos.

Na seção 2 descrevemos o problema do particionamento em grafos e o que nos levou a fazer o trabalho. Na seção 3 é mostrado o objetivo. Na seção 4 são mostrados os algoritmos propostos. A seção 5 contém os resultados obtidos em cada proposta. E por fim, na seção 6, a conclusão do trabalho.

2. Problema e Motivação

Dado um numero $k \in N$ e $k > 1$ e um grafo $G = (V, E)$ não direcionado sem pesos negativos nas arestas, $w : E \rightarrow R > 0$, o problema do particionamento em grafos requisita uma partição P de V com clusters de nós $C_i = (V1, ..., Vk)$:

$$1.\ V_1 \cup \cdots \cup V_k = V$$
$$2.\ V_i \cap V_j = \emptyset \ \ \forall i \neq j$$

[Buluc et al. 2013]

A ideia é maximizar ou minimizar uma função objetivo escolhida. A função objetivo usada neste trabalho é:

$$\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \sum_{u \in C_i} \sum_{v \in C_j} dG(u, v)$$

Onde $dG(u, v)$ é a distancia entre u e v em G A motivação do trabalho é melhorar o gerenciamento do restaurante universitário, e um sistema de agendamento utilizando bons resultados sobre o problema do particionamento é um grande salto nesse caminho, já que quanto melhor a partição, maior é a afinidade geral do grupo agendado.

Como a aresta representa afinidade entre dois vertices, os grafos retratados nesse trabalho tem arestas com pesos unitários.

3. Objetivo

O objetivo do trabalho é encontrar um particionamento com o resultado o mais próximo possível do particionamento ótimo, para assim conseguir um agendamento que agrade o grupo de pessoas que usufrui do restaurante, e uma boa distribuição na quantidade de refeições feitas.

4. Propostas

O problema foi abordado utilizando três estrategias diferentes

4.1. Estrategia um

Na primeira estrategia, os nós com maiores graus são escolhidos como representantes, cada um é selecionado para um cluster diferente, e então cada cluster é preenchido por completo, seguindo a ordem, do primeiro ao ultimo, até que todos os nós estejam alocados em um cluster.

Algorithm 1: Strategy One

```
listNodes ← inicializeListaNodes(graph) {O(V)}
s1 ← n % k
length1 ← ceil(n / k)
length2 ← floor(n / k)
distMatrix ← BfsAllPairs(graph) {O(V^2 + V * E)}
for i from 0 to k do
   node ← getHighestDegreeNode(listNodes) {O(V)}
   cluster[i].append(node) {O(1)}
end for
for i from 1 to s1 do
   reference ← cluster[i].getFirst() {O(1)}
   while cluster[i].size() < length1 do
      node ← getClosestNode(reference, listNodes, distMatrix) {O(V)}
      cluster[i].append(node)
      listNodes.delete(node) {O(1)}
   end while
end for
for i from s1+1 to k do
   reference ← cluster[i].getFirst()
   while cluster[i].size() < length2 do
      node ← getClosestNode(reference, listNodes, distMatrix)
      cluster[i].append(node)
      listNodes.delete(node)
   end while
end for
```

4.2. Estrategia dois

Na segunda estratégia, os nós com maiores graus são escolhidos como representantes, cada um é selecionado para um cluster diferente, e então cada cluster seleciona o melhor canditato para ele, por iteração, até que todos os nós estejam alocados em um cluster.

Algorithm 2: Strategy Two

```
listNodes ← inicializeListaNodes(graph) {O(V)}
s1 ← n % k
length1 ← ceil(n / k)
length2 ← floor(n / k)
distMatrix ← BfsAllPairs(graph) {O(V^2 + V * E)}
for i from 0 to k do
   node ← getHighestDegreeNode(listNodes) {O(V)}
   cluster[i].append(node) {O(1)}
end for
while list not empty do
   reference ← cluster[i].getFront()
   node ← getClosestNode(reference, listNodes, distMatrix) {O(V)}
   cluster[i].append(node) {O(1)}
   list.delete(node) {O(V)}
   i ← i + 1
   if i == k then
      i ← 0
   end if
end while
```

4.3. Algoritmo Genético

O algoritmo genético [Holland 1992] se baseia na ideia da seleção natural de Darwin, onde indivíduos com melhor adaptação são os mais capacitados para exercer determinada função, e invididuos melhores são formados por meio de mutações genéticas e reprodução entre os individuos com melhor capacitados. Como o problema pede para maximizar a função de distancia entre os clusters, foi usado o algoritmo genético na tentativa de maximizar.

Algorithm 3: Genetic Algorithm

```
initializePopulation()
evaluatePopulation()
while !stopCondition do
   select the best fit individuals to reproduce
   create new individuals based on the selected individuals through mutations
   evaluate new individuals
   replace least fit individuals from population with new best fit individuals
end while
return best individual
```

4.4. Análise de Complexidade

As estratégias um e dois possuem a mesma complexidade, pois as duas estrategias definem k vertices como representantes, e preenchem seus clusters baseados na distancia dos vertices restantes aos representantes. A complexidade dessas duas estratégias é $O(V^2 + V * E)$, já que o input k sempre será menor que os vertices, a complexidade da busca em largura realizada em cada vertice, para obter a menor distancia de todos para todos é a maior do algoritmo.

A estrutura do cromossomo no algoritmo genético é um vetor contendo todos os vertices, portanto acessar um vértice leva tempo constante. A quantidade da populaçao, quantidade de filhos gerados por iteração e o numero de iterações também é constante, portanto a complexidade recai sobre a função de adaptação e o operador de mutação escolhidos.

Na mutação, clusters diferentes são escolhidos aleatoriamente e ocorre uma troca de vertices entre eles, portanto leva tempo constante. A criação do filho precisa que um vetor seja copiado para outro, portanto custa $O(V)$. A função de adaptação escolhida foi a função objetivo apresentada na seção 2, onde é preciso calcular a distancia de cada vertice para todos os outros que não estejam em seu cluster. Seu custo é $O(V^2)$

5. Experimentos

Os testes foram executados em uma máquina com as seguintes especificações:
Processador: Intel(R)Core(TM)i5-4690 CPU @ 3.50GHz
Compilador: Gcc 8.2
Capacidade da Cache: 6MB Cache L3
Distribuição Linux: Arch Linux

O conjunto de dados utilizado no teste foram os grafos dados como exemplo na descrição da atividade. Dentre eles, as instancias variam em quantidade de vertice, de 100 até 300, e em quantidade de arestas, de 471 até 14834.

Como o genético não gera sempre a mesma solução, o resultado foi selecionado pela média de 10 execuções, retirando a maior e a menor solução antes do calculo da média. O tempo médio foi obtido da mesma maneira.

Imagens de como ficaram particionadas algumas instancias utilizando o algoritmo genético:

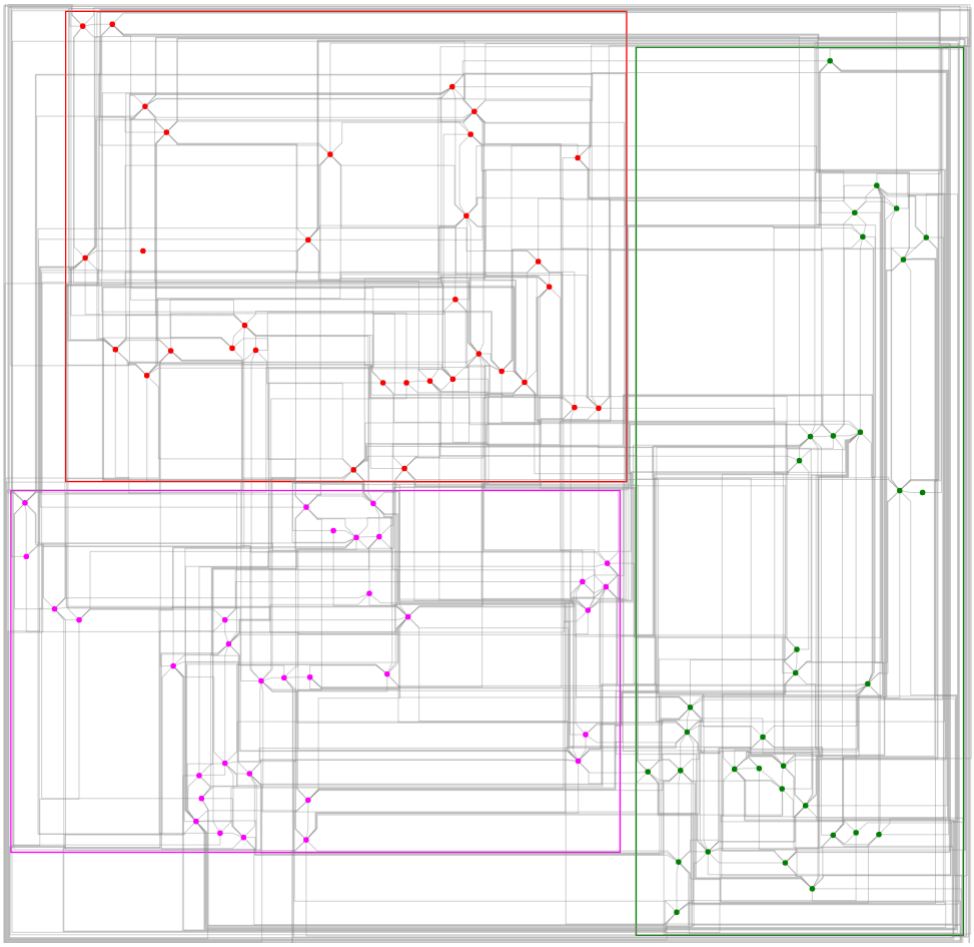

Figura 1. g1, vertices separados, k = 3

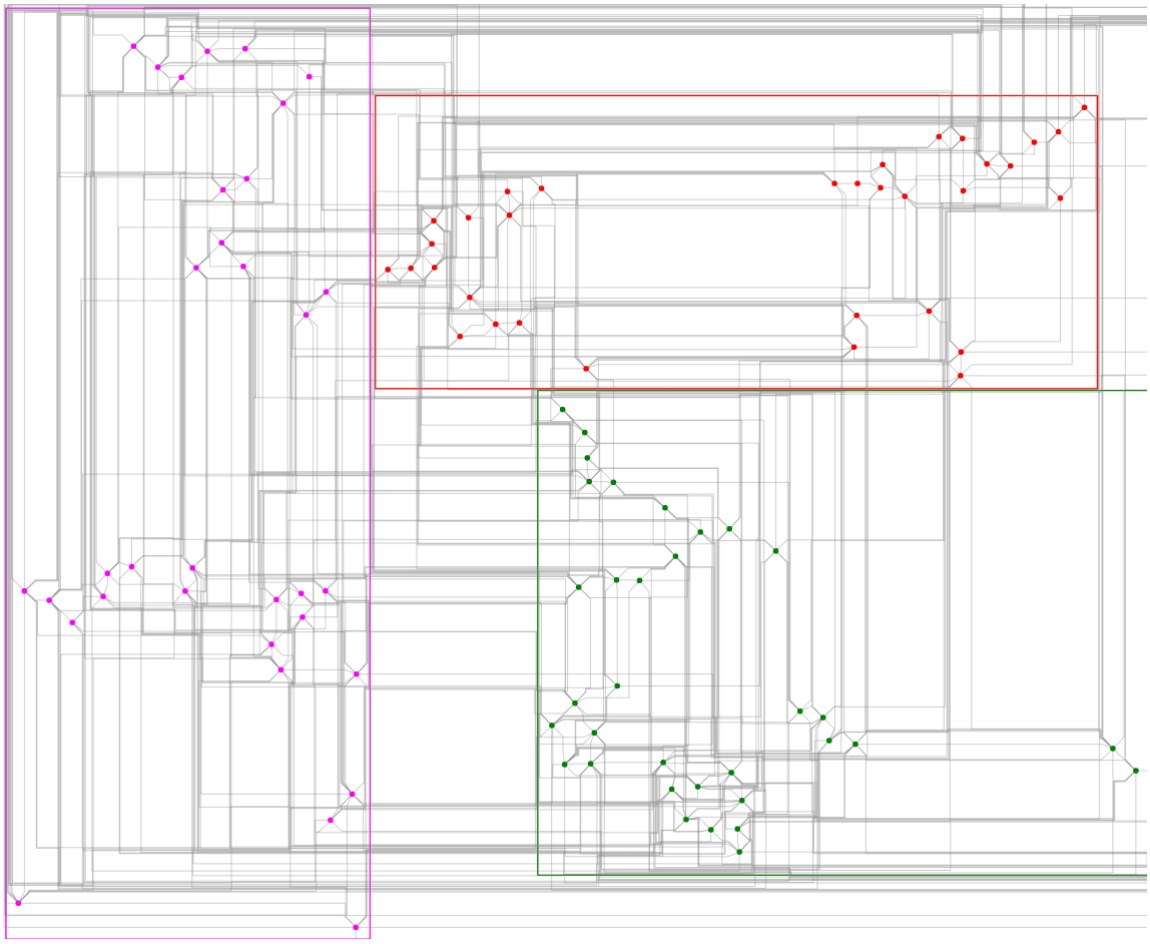

Figura 2. g2, vertices separados, k = 3

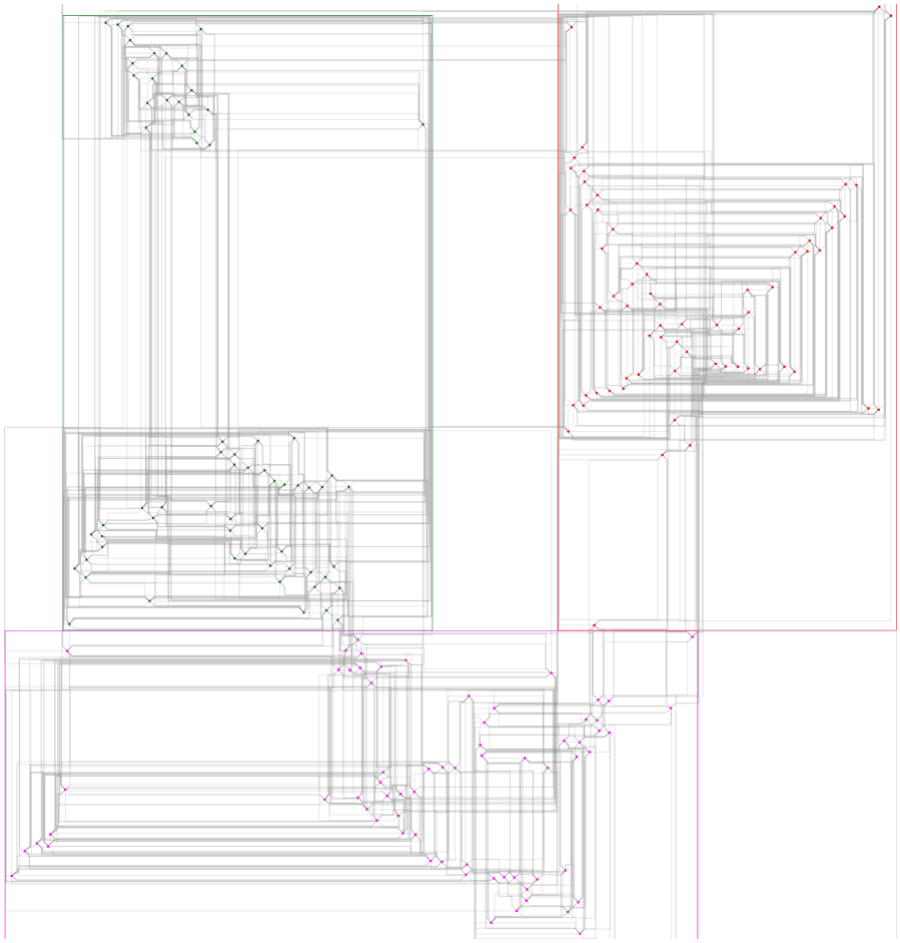

Figura 3. c-fat200-1, vertices separados, k = 3

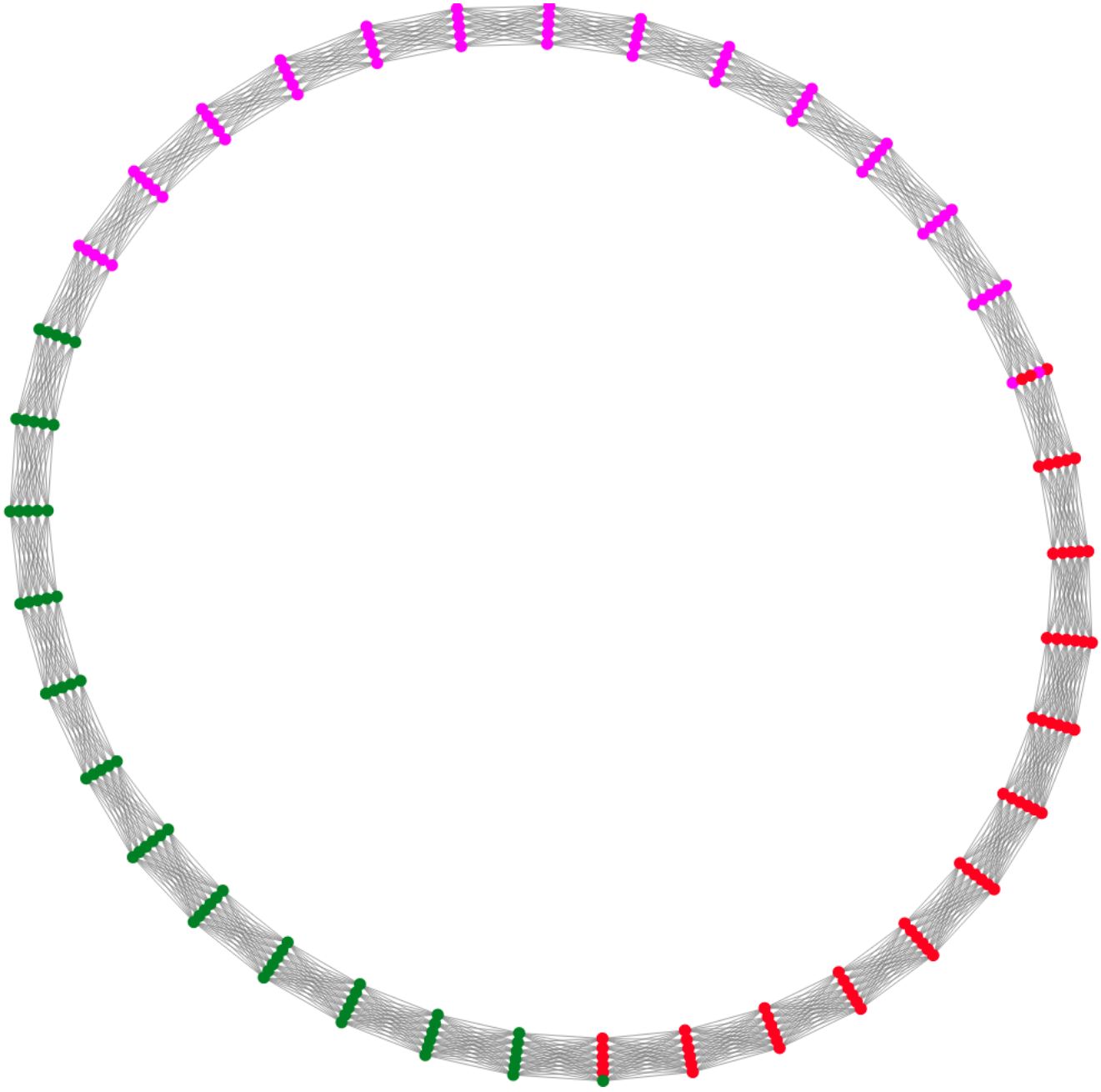

Figura 4. c-fat200-1, vertices não separados, k = 3

Tabelas com resultados obtidos:

Instância	Vértices	Arestas	Est.Um	Est.Dois	Genético
g1	100	471	5767	5802	6022
g2	100	499	5648	5612	5905
c-fat200-1	200	1534	121632	111323	122540
p_hat300-1	300	10933	39607	39675	40206
C125.9	125	6963	4324	4318	4439
brock200_1	200	14834	12524	12594	12964

Tabela 1. Somatórios obtidos, K = 2

Instância	Vértices	Arestas	Est.Um	Est.Dois	Genético
g1	100	471	7661	7752	7956
g2	100	499	7536	7571	7792
c-fat200-1	200	1534	145193	130668	156748
p_hat300-1	300	10933	52807	52837	53516
C125.9	125	6963	5745	5745	5889
brock200_1	200	14834	16726	16755	17201

Tabela 2. Somatórios obtidos, K = 3

Instância	Vértices	Arestas	Est.Um	Est.Dois	Genético
g1	100	471	8669	8670	8913
g2	100	499	8477	8510	8706
c-fat200-1	200	1534	157407	142974	169018
p_hat300-1	300	10933	59404	59384	60139
C125.9	125	6963	6452	6453	6603
brock200_1	200	14834	18824	18896	19282

Tabela 3. Somatórios obtidos, K = 4

Instância	Vértices	Arestas	Est.Um	Est.Dois	Genético
g1	100	471	0.001	0.019	2.548
g2	100	499	0.001	0.019	2.240
c-fat200-1	200	1534	0.002	0.067	8.452
p_hat300-1	300	10933	0.056	0.277	20.709
C125.9	125	6963	0.013	0.135	3.569
brock200_1	200	14834	0.048	0.253	8.983

Tabela 4. Tempo médio de execução em segundos, K = 3

6. Conclusões

O problema apresentado nesse projeto consiste em maximizar a função de distância entre os clusters formados. Fez-se uso de duas estrátegias um pouco semelhantes e de uma um pouco mais distinta para tentativa de melhorias. Perante os resultados encontrados constata-se que o algoritmo genético foi superior em todas as instancias. Como não há um exemplo de melhor partição ou o resultado ótimo desse problema nas instancias, não é possível avaliar a margem de erro entre o resultado médio do algoritmo genético e o resultado ótimo.

Referências

Buluc, A., Meyerhenke, H., Safro, I., Sanders, P., and Schulz, C. (2013). Recent advances in graph partitioning. In *Recent Advances in Graph Partitioning*, pages 2–3.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. MIT Press, 1992 edition.