UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FELIPE ZORZO PEREIRA

Simulated Annealing para o Problema dos Grupos Balanceados de Maior Valor

Projeto apresentado com o objetivo de melhor entender a meta-heurística Simulated Annealing por meio de testes sobre o problema dos grupos balanceados de maior valor.

Orientador: Prof. Dr. Marcus Ritt

Porto Alegre
2018

SUMÁRIO

1 INTRODUÇÃO

O objetivo deste trabalho foi analisar os resultados obtidos pela aplicação de uma meta-heurística ao problema dos grupos balanceados de maior valor. A meta-heurística escolhida foi o Simulated Annealing, cujos detalhes serão minuciados nas próximas seções.

A definição do problema dos grupos balanceados de maior valor é: dado um grafo não-direcionado G = (V, A) com valor $d_{u,v} \geq 0$ da aresta entre os vértices u,v ∈ V, pesos p_v para cada vértice v ∈ V, g grupos com limites L (inferior) e U (superior) para o peso total do grupo, maximiza o valor total das arestas entre vértices do mesmo grupo.

Ok

2 FORMULAÇÃO DO PROGRAMA INTEIRO

Variáveis: A variável $x_{i,j}$ é igual a 1 se vértice i ∈ [n] está no grupo j ∈ [g]; caso contrário, $x_{i,j}$ é igual 0. A variável $c_{i,k,j}$, por sua vez, é igual a 1 se vértices i ∈ [n] e k ∈ [n] estão no grupo j ∈ [g]; caso contrário, é igual a 0. A letra g denota o número de grupos da instância do problema, enquanto n o número de vértices.

Função Objetivo:

$$maximiza \sum_{i\in[n]} \sum_{k\in[n]} \sum_{j\in[g]} c_{i,k,j} d_{i,k}$$

Restrições:

$$\sum_{i\in[n]} x_{i,j} p_i \leq U_j, \forall j \in [g], \quad (1)$$

$$\sum_{i\in[n]} x_{i,j} p_i \geq L_j, \forall j \in [g], \quad (2)$$

$$\sum_{j\in[g]} x_{i,j} = 1, \forall i \in [n], \quad (3)$$

$$c_{i,k,j} \leq \frac{x_{i,j} + x_{k,j}}{2}, \forall\, i \in [n], k \in [n], j \in [g], \quad (4)$$

$$c_{i,k,j} \in \{0,1\}, \forall\, i \in [n], k \in [n], j \in [g], \quad (5)$$

$$x_{i,j} \in \{0,1\}, \forall\, i \in [n], j \in [g]. \quad (6)$$

A restrição (1) garante que o peso de cada grupo não ultrapasse o seu limite superior; da mesma forma, a restrição (2) garante que o peso de cada grupo não seja menor que seu limite inferior.

A restrição (3) faz com que cada vértice esteja no máximo em um grupo.

A restrição (4) obriga $c_{i,k,j}$ a seguir sua definição: ela só poderá ser igual a 1 se tanto o vértice i ∈ [n] quanto o vértice k ∈ [n] estiverem no grupo j.

Ressalta-se que p_i denota o peso do vértice i ∈ [n], L_j e U_j o peso total mínimo e máximo, respectivamente, do grupo j ∈ [g] e $d_{i,k}$ o valor da aresta entre os vértices i ∈ [n] e k ∈ [n]. Caso não exista aresta entre tais vértices, considera-se que $d_{i,k} = 0$.

Bom.

3 ELEMENTOS E PARÂMETROS DO SIMULATED ANNEALING

O Simulated Annealing tenta fugir de mínimos e máximos locais. Para tanto, soluções vizinhas piores que a solução atual são aceitas com probabilidade $e^{delta/T}$, com delta sendo um número real que representa a diferença entre a qualidade do vizinho gerado e a qualidade da solução atual. Como o problema dos grupos balanceados de maior valor é de maximização, delta é negativo sempre que o vizinho for pior que a solução atual.

Tendo em vista isto, quanto maior T, que representa a temperatura atual do algoritmo, maior será a chance de um vizinho pior ser aceito. O valor de T é diminuído ao longo da execução do algoritmo ao ser multiplicado por um valor real entre 0 e 1, chamado de fator de resfriamento. Com isso, inicialmente a probabilidade de um vizinho pior ser aceito é alta, mas diminui ao longo da execução do algoritmo.

Dadas essas informações, os parâmetros necessários para executar o Simulated Annealing são:

r: fator de resfriamento do algoritmo, número real entre 0 e 1.

I: número de vizinhos gerados em uma mesma temperatura do algoritmo, inteiro maior ou igual a 1.

k: número de trocas a serem realizados pelo k-change durante a criação de um vizinho (conforme será descrito na seção 4.2), inteiro maior ou igual a 1.

pf: probabilidade final, número real entre 0 e 1. Probabilidade mínima desejada de se realizar um movimento para um vizinho pior.

pi: probabilidade inicial, número real entre 0 e 1. Probabilidade desejada de o algoritmo realizar um movimento para um vizinho pior durante sua temperatura inicial.

nsi: número de soluções iniciais geradas antes de passar a melhor delas para o Simulated Annealing, número inteiro maior ou igual a 1.

Ok

4 O ALGORITMO ADAPTADO AO PROBLEMA

```
simulatedAnnealing(r, I, pf, pi, nsi, n, g, A, L, U, p, k):
 1:     counter ← 0
 2:     s ← CriaSoluçãoInicial(n, g, L, U, p, nsi)
 3:     T ← EncontraTemperaturaInicial(s, r, I, pi, n, g, A, L, U, p, k)
 4:     while counter < 10 do
 5:         movesTried ← 0
 6:         movesSucceded ← 0
 7:         for i in 1 to I do
 8:             s' ← CriaVizinho(g, n, s, L, U, p, k)
 9:             if s' ≠ s then
10:                 delta ← ValorSolução(s', g, A) - ValorSolução(s, g, A)
11:                 exponent ← delta/T
12:             else
13:                 delta ← -1
14:                 exponent ← -∞
15:             end if
16:             if delta ≥ 0 then
17:                 s ← s'
18:                 counter ← 0
19:             else
20:                 movesTried ← movesTried + 1
21:                 if GeraRandômico(0, 1) < e^exponent then
22:                     s ← s'
23:                     movesSucceded ← movesSucceded + 1
24:                 end if
25:             end if
26:         end for
27:         if movesTried > 0 then
28:             if (movesSucceded / movesTried) < pf then
29:                 counter ← counter + 1
30:             end if
31:         end if
32:         T ← r*T
33:     end while
34:     return s
```

Ok

4.1 Entradas da instância do problema

Além dos parâmetros do Simulated Annealing em si, descritos na seção 3, o algoritmo também recebe algumas entradas decorrentes do problema a ser resolvido:

A: valores das arestas entre os vértices.

L: limites inferiores do peso de cada grupo.

U: limites superiores do peso de cada grupo.

p: pesos dos vértices.

n: número de vértices.

g: número de grupos

4.2 Vizinhança

A vizinhança implementada foi a k-change. Para criar um novo vizinho aleatório, k vértices aleatórios são mudados de grupo; os grupos que perdem ou ganham vértices são, também, escolhidos de forma aleatória. Se as mudanças gerarem uma solução factível, ela é retornada; caso contrário, o processo se repete por um número fixo de iterações (valor máximo entre 10000 e n^2). Caso um vizinho factível não seja criado ao fim das iterações, a solução que tentava-se perturbar é retornada.

Ok, mas talvez grande demais.

4.3 Representação de uma solução

É representada por uma lista de listas. Cada uma das listas representa um grupo da instância do problema; cada elemento da lista é o índice de um dos vértices que está no grupo representado por esta lista.

Ok

4.4 Solução inicial

É feita de forma aleatória. Adiciona-se vértices aleatórios da instância do problema a um grupo até que o peso desse grupo seja maior ou igual a seu limite inferior. Após este passo ser realizado para todos os grupos, se ainda há vértices sem grupo, eles são adicionados a algum. Após todos os vértices terem sido adicionados a um grupo, a solução é testada para confirmar sua factibilidade. Se é factível, a solução é retornada; caso contrário, a distribuição de vértices é feita novamente. O algoritmo tenta criar uma solução inicial por um número fixo de iterações (valor máximo entre 10000 e n^2); se tal número é extrapolado, o Simulated Annealing é cancelado e retorna uma mensagem de erro.

Ok

4.5 Temperatura inicial

O próprio Simulated Annealing é executado para encontrar uma temperatura inicial que aceite um movimento para um vizinho pior com probabilidade próxima à probabilidade inicial definida.

Bom

4.6 Critério de parada

O critério é definido pela probabilidade final utilizada. Quando, durante dez temperaturas seguidas (contadas pela variável *counter*), realizar um movimento para um vizinho pior ocorreu com probabilidade menor que a probabilidade final, o algoritmo é considerado resfriado e, portanto, parado.

Bom

5 IMPLEMENTAÇÃO

5.1 Plataforma de implementação

O algoritmo foi implementado e testado em uma máquina com sistema operacional Windows 10 Enterprise (64-bit), 16GB de memória principal e processador Intel(R) Core(TM) i7-7700K com 4 núcleos físicos de 4.2GHz, cada um com 32KB de cache L1 e 256KB de cache L2. A linguagem de programação utilizada foi Python 3.6.3.

Ok

5.2 Estruturas de dados utilizadas

Para representar as arestas existentes entre vértices optou-se por utilizar uma matriz de adjacência não-espelhada.

Os limites inferiores e superiores são armazenados em listas. O elemento de índice j dessas listas contém o limite inferior/superior do grupo de índice j.

Os pesos dos vértices, assim como os limites, são armazenados em uma lista. O elemento de índice i contém o peso do vértice de índice i.

Ok

5.3 Impossibilidade de criar um vizinho

Caso a criação de vizinhos não consiga gerar um vizinho factível para uma solução, essa solução é retornada sem qualquer alteração, como citado na seção 4.2. Isso faz com que *delta* seja igual a 0 e que, consequentemente, o contador utilizado para definir quando o algoritmo deve ser parado seja zerado. Esse comportamento é indesejado, uma vez que o algoritmo poderia ficar preso em uma solução sem vizinhos.

Para solucionar esse problema, optou-se por, quando o vizinho devolvido for igual à solução atual, pôr um valor negativo em *delta* e fazer *exponent* tender a menos infinito. Isso garante que *counter* não seja zerado e que a taxa de aceitação de movimentos ruins decaia. Com isso, se uma solução não tiver vizinhos factíveis, a taxa de aceitação de movimentos ruins decai até ser menor que a probabilidade final; isso se repete durante as próximas nove temperaturas. O algoritmo é, então, parado.

Ok

6 TESTES DE PARÂMETROS

Para os seguintes testes, variou-se cada parâmetro de entrada separadamente, para determinar qual o valor ideal de cada um. Os valores padrão para os parâmetros que não estavam sendo testados são como segue: $r = 0.99$; $I = 2500$; $pf = 0.05$; $pi = 0.85$; $nsi = 10000$; $k = 1$.

Cada um dos parâmetros foi testado para a instância gbmv240_01 e cada teste foi rodado três vezes, utilizando sementes aleatórias. O tempo de execução de cada teste foi medido. Fez-se média simples dos valores da solução final e do tempo de execução de cada teste.

6.1 Teste do fator de resfriamento (parâmetro *r*)

Os valores testados de *r* foram 0.99, 0.95 e 0.90. As figuras 6.1.1 e 6.1.2 mostram os resultados obtidos.

Figura 6.1.1 – Valor da solução encontrada em relação à variação do fator de resfriamento

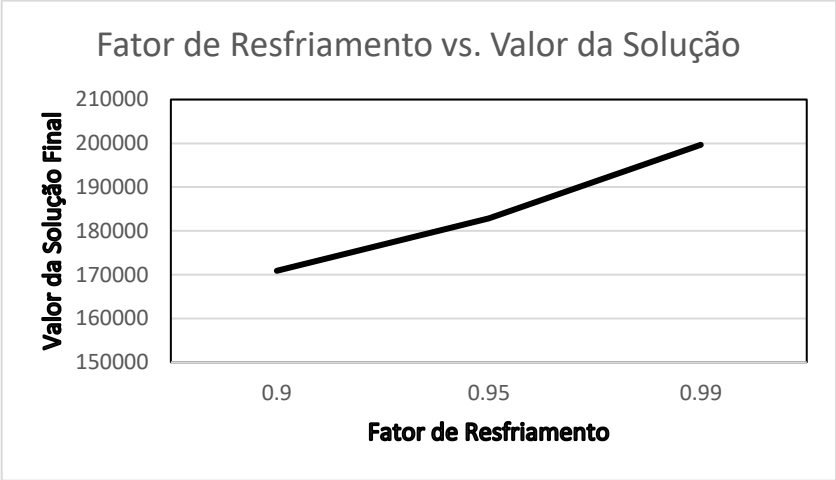
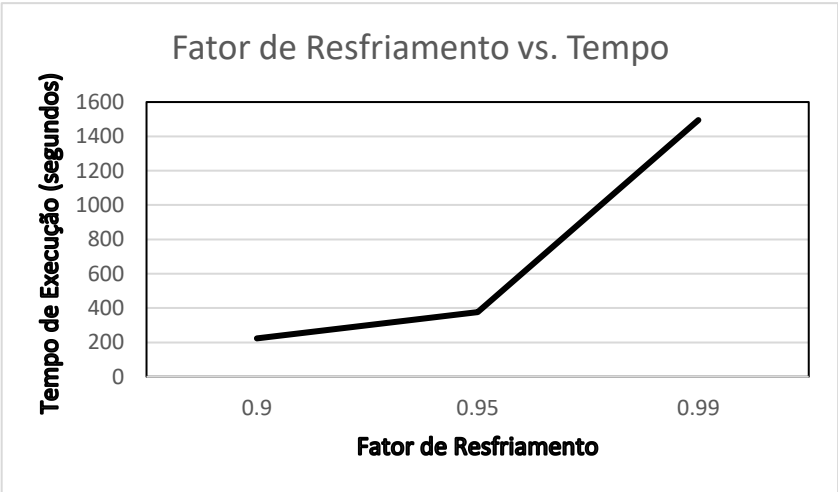


Figura 6.1.2 – Tempo de execução em relação à variação do fator de resfriamento



Observa-se que o valor da solução final aumenta de maneira quase linear conforme o fator de resfriamento é aumentado. O tempo de execução quase dobra ao passar de $r = 0.90$ para $r = 0.95$ e, ao utilizar 0.99 como o valor de r, o tempo de execução aumenta drasticamente.

Apesar do aumento no tempo de execução, conclui-se que é aceitável utilizar r próximo a 1, devido ao aumento da qualidade da solução final.

Ok

6.2 Teste do número de vizinhos gerados por temperatura (parâmetro *I*)

Testou-se *I* com os valores 50, 250, 500, 1000 e 2500. As figuras 6.2.1 e 6.2.2 mostram os resultados.

Figura 6.2.1 – Valor da solução encontrada em relação à variação de *I*

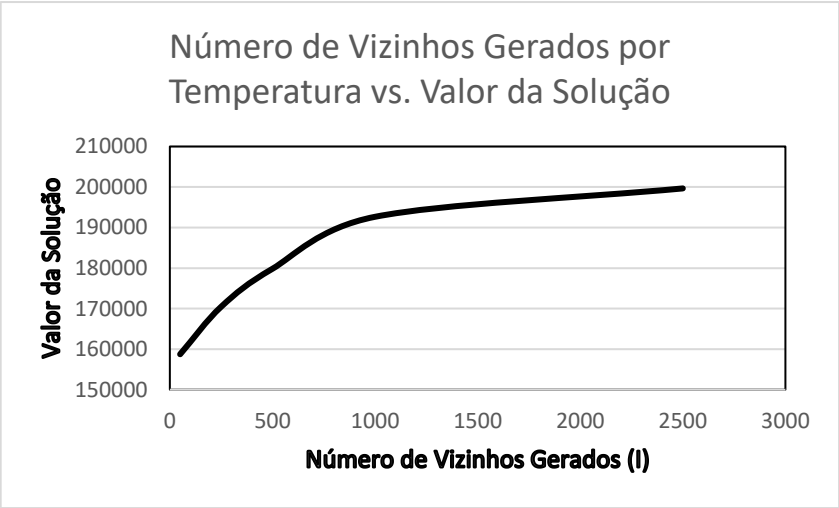
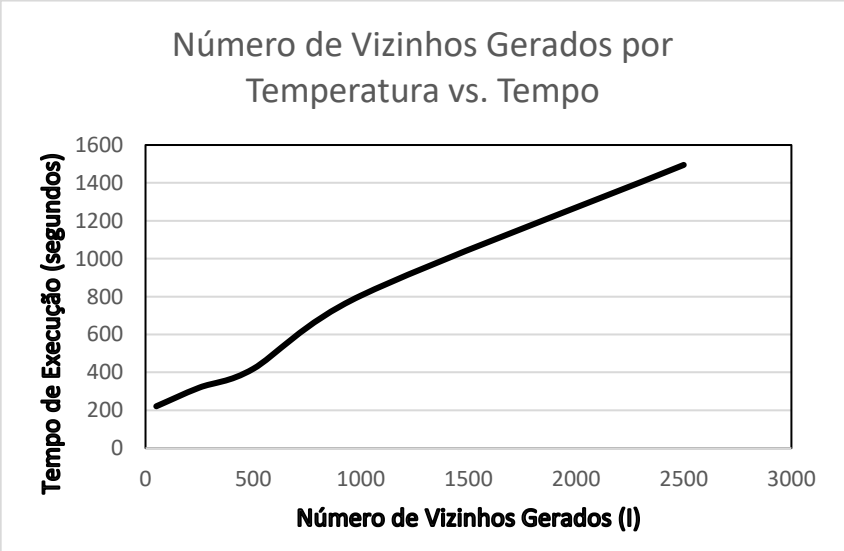


Figura 6.2.2 – Tempo de execução em relação à variação de *I*



Este parâmetro é o que traz as maiores melhorias quanto ao valor absoluto da solução final encontrada. Observa-se que o valor da solução final cresce de forma logarítmica com o aumento do número de vizinhos criados por temperatura.

Essa melhoria trouxe consigo um impacto negativo no tempo de execução do algoritmo, que aumenta linearmente. Conclui-se, então, que $I = 2500$ é um valor satisfatório, uma vez que garante uma melhoria significativa no valor da solução final sem impactar de forma muito negativa o tempo de execução.

Ok

6.3 Teste da probabilidade inicial (parâmetro *pi*)

Para este parâmetro foram realizados quatro testes. Os valores testados foram 0.85, 0.90, 0.95 e 1. As figuras 6.3.1 e 6.3.2 demonstram os resultados atingidos.

Figura 6.3.1 – Valor da solução encontrada em relação à variação de *pi*

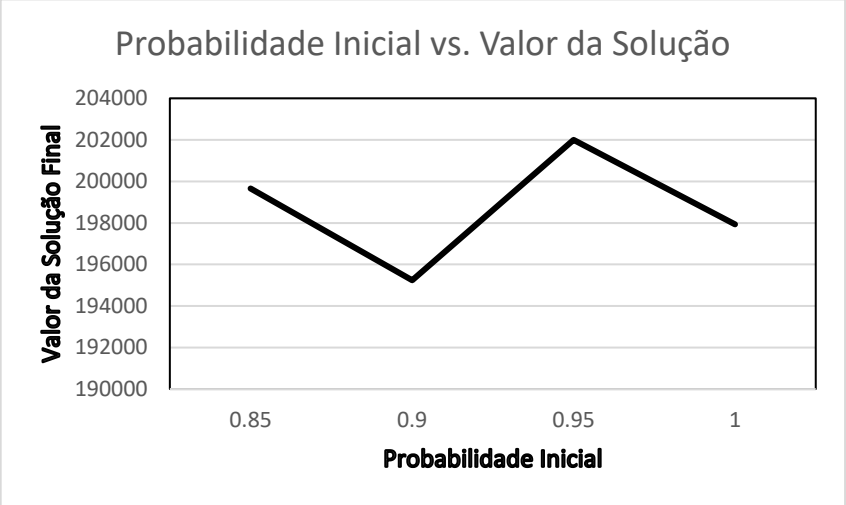
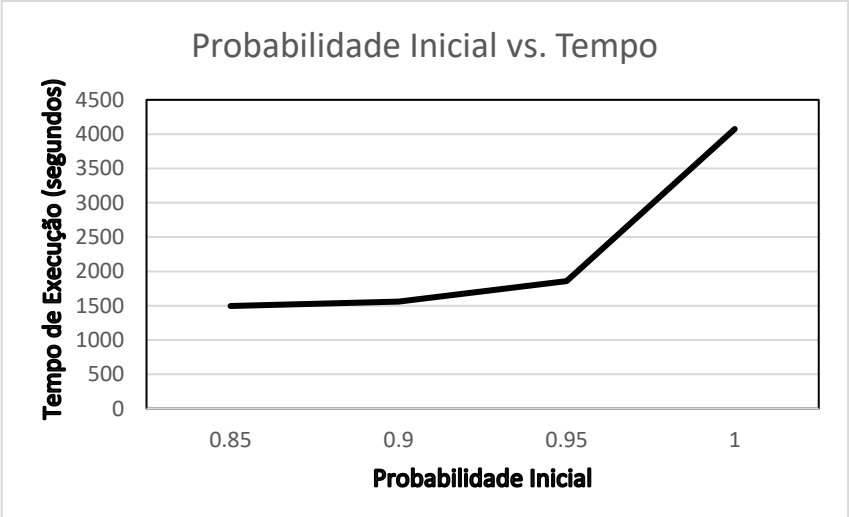


Figura 6.3.2 – Tempo de execução em relação à variação de *pi*



Conforme *pi* aumenta, os valores das soluções finais variam de forma não-uniforme: em um momento o valor melhora, no outro piora. O tempo de execução, por sua vez, apresenta leves aumentos quando *pi* aumenta; a exceção é quando se passa de *pi* = 0.95 para *pi* = 1, em que o tempo de execução mais que dobra.

Desta forma, conclui-se que o valor ideal da probabilidade inicial é 0.95, pois o tempo de execução não aumenta muito em relação a probabilidades iniciais menores, mas o valor da solução final é maior.

Ok

6.4 Teste da probabilidade final (parâmetro *pf*)

Testou-se a probabilidade final com os valores 0.01, 0.05, 0.10 e 0.15. As figuras 6.4.1 e 6.4.2 mostram os resultados.

Figura 6.4.1 – Valor da solução encontrada em relação à variação de *pf*

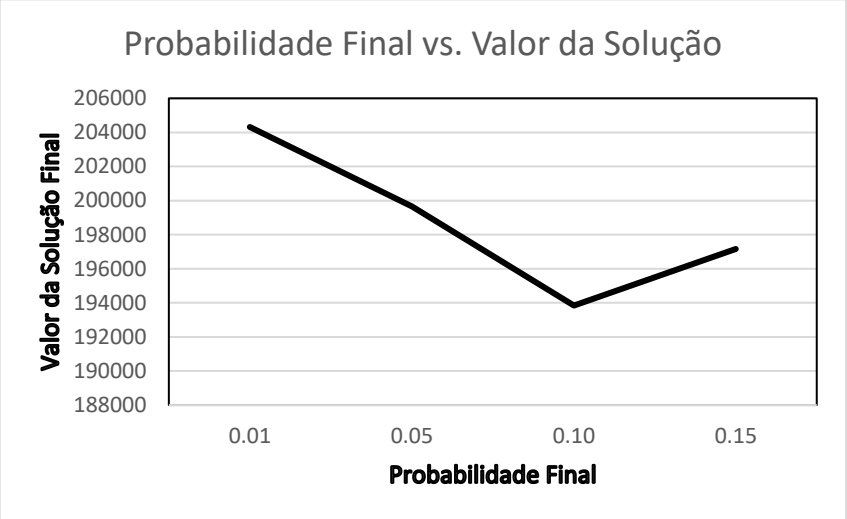
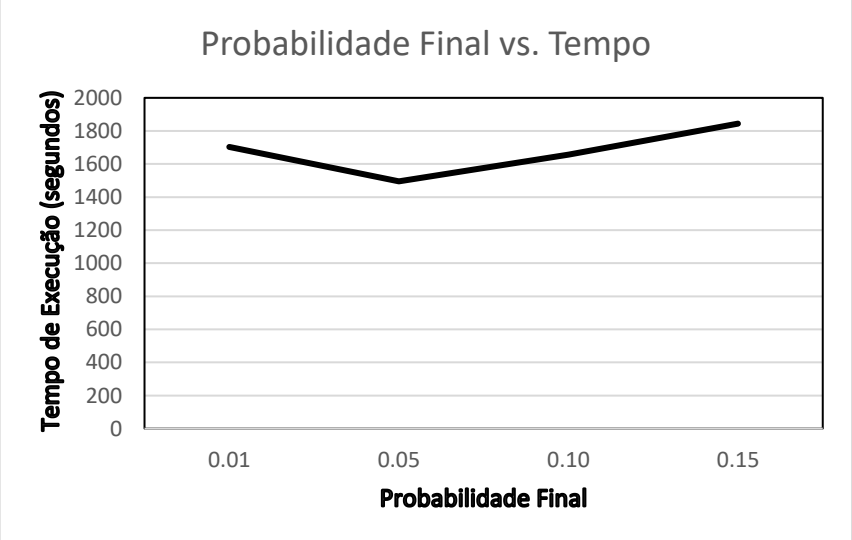


Figura 6.4.2 – Tempo de execução em relação à variação de *pf*



Observou-se que, quanto maior a probabilidade final, menores os valores, com exceção do valor da solução final encontrada com *pf* = 0.15, que é maior que o valor da solução encontrada com *pf* = 0.10. Os tempos de execução, por outro lado, se mostram pouco variáveis.

Analisando os resultados, conclui-se que o ideal é escolher para a probabilidade final um valor próximo de 0, já que esta praticamente não influi no tempo gasto, mas melhora os valores das soluções finais.

Ok

6.5 Teste do número de soluções iniciais geradas (parâmetro *nsi*)

Testou-se a geração de 1, 100, 1000 e 10000 soluções iniciais, das quais a melhor era passada ao Simulated Annealing. As figuras 6.5.1 e 6.5.2 mostram os resultados.

Figura 6.5.1 – Valor da solução encontrada em relação à variação de *nsi*

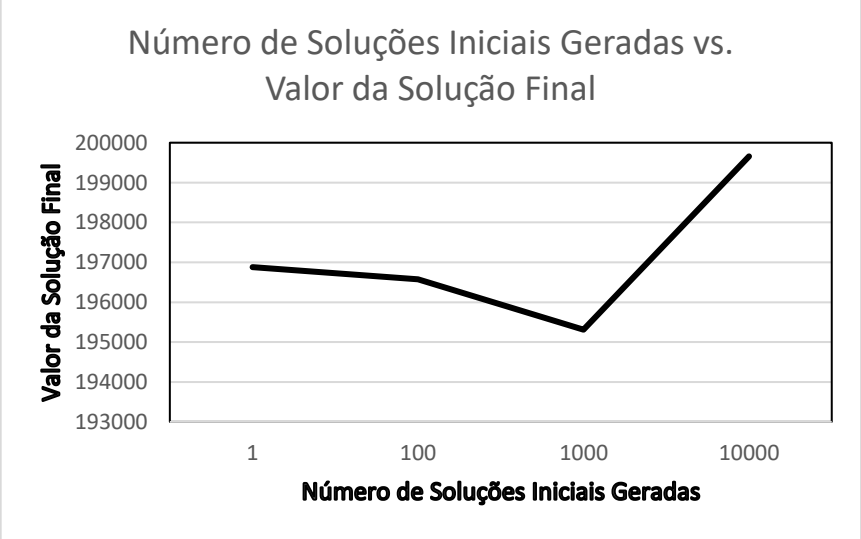
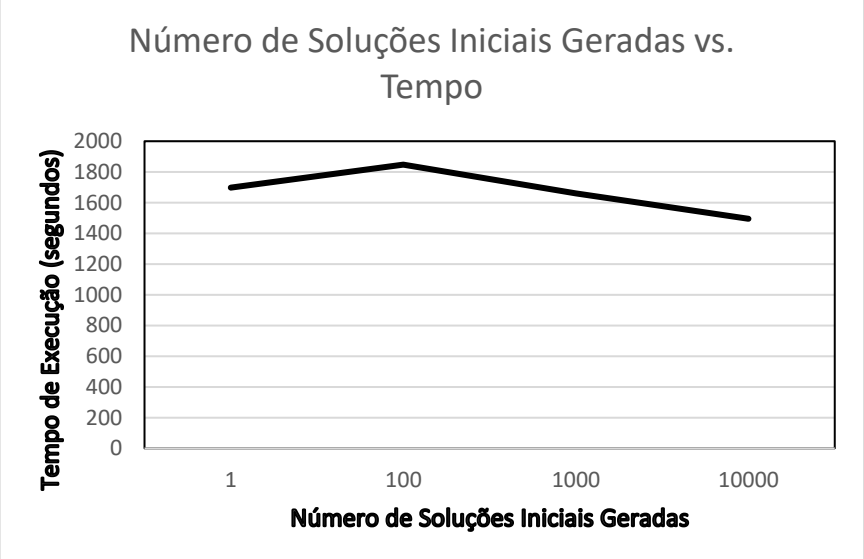
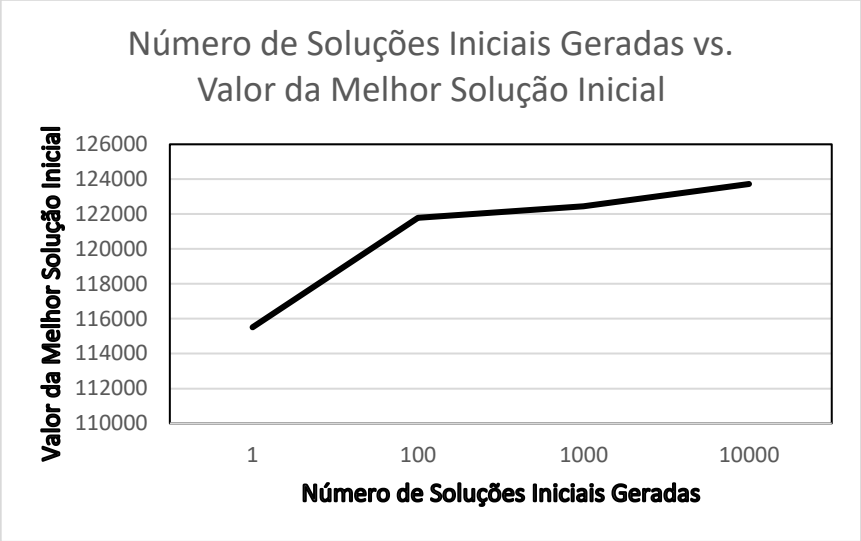


Figura 6.5.2 – Tempo de execução em relação à variação de *nsi*



A figura 6.5.3 mostra o valor médio da melhor solução inicial encontrada para cada valor de *nsi* testado.

Figura 6.5.3 – Valor da solução encontrada em relação à variação de *nsi*



Analisando os resultados, percebeu-se que o número de soluções iniciais geradas e, consequentemente, a qualidade da solução inicial, não é um parâmetro muito significativo para o Simulated Annealing: gerar apenas uma solução inicial, de valor próximo a 115000, levou a uma solução final melhor do que quando geradas 100 ou 1000 soluções iniciais, das quais a melhor tinha valor próximo a 122000. Além disso, notou-se que a qualidade das soluções iniciais fica bastante estagnada a partir de *nsi* = 100.

Ademais, o número de soluções iniciais geradas claramente não influi significantemente no tempo de execução do algoritmo, já que ao se criar 10000 soluções iniciais o algoritmo executou em menos tempo que quando se criou apenas uma solução inicial.

Conclui-se que qualquer número de soluções iniciais é aceitável; optou-se por utilizar, entretanto, 10000, uma vez que foi o que gerou melhores resultados em menor tempo.

Ok

6.6 Teste do número de trocas da vizinhança (parâmetro *k*)

Testou-se *k* com os valores 1, 2 e 4. As figuras 6.6.1 e 6.6.2 mostram os resultados.

Figura 6.6.1 – Valor da solução encontrada em relação à variação de *k*

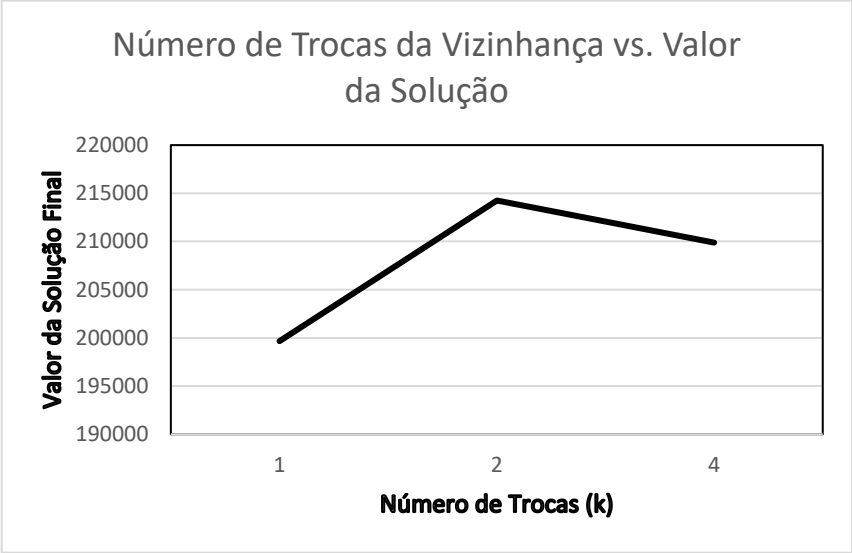
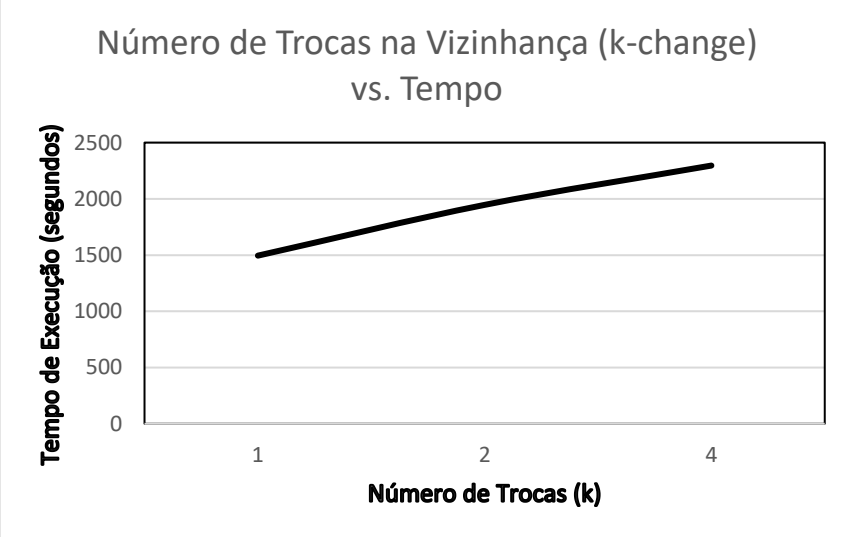


Figura 6.6.2 – Tempo de execução em relação à variação de *k*



Alterações neste parâmetro levaram o algoritmo a atingir um valor de 210000 para a instância gbmv240_01 pela primeira vez, se mostrando bastante significativo.

Observou-se que, conforme *k* aumenta, o tempo de execução também aumenta. Os valores, por sua vez, aumentam ao se passar de $k = 1$ para $k = 2$, mas voltam a descer ao se passar de $k = 2$ para $k = 4$. Deve-se cuidar para não utilizar *k* muito grande, uma vez que, com *k* tendendo ao infinito, a geração de vizinhos se torna um passeio aleatório sobre o grafo de possíveis soluções do problema.

Conclui-se, portanto, que utilizar $k = 2$ é a alternativa ideal, já que a penalidade no tempo de execução não é tão alta e o ganho no valor da solução final é o maior.

Ok

7 TESTES DAS INSTÂNCIAS

O Simulated Annealing foi testado com dez instâncias do problema dos grupos balanceados de maior valor, disponíveis em http://www.inf.ufrgs.br/~mrpritt/oc/gbmv.zip. Os valores utilizados para os parâmetros, escolhidos com base nos melhores valores encontrados nos testes da seção 6, foram: $r = 0.99$, $I = 2500$, $pf = 0.01$, $pi = 0.95$, $nsi = 10000$ e $k = 2$.

Cada instância de teste foi executada até o algoritmo se considerar resfriado e parar, de acordo com o critério de parada definido na seção 4.5. Os resultados são mostrados na Tabela 7.1 e na Tabela 7.2. As mesmas instâncias foram testadas utilizando-se o solver GLPK com limite de tempo de uma hora. Durante esse tempo o solver não conseguiu encontrar solução inteira factível para nenhuma das instâncias; logo, seus resultados foram omitidos das tabelas.

Tabela 7.1 – Desvio percentual da solução final em relação à inicial

Instância	Solução Inicial	Solução Final	Desvio para Sol. Inicial (%)	Tempo de Execução (segundos)	Semente
gbmv240_01	124660.52	214448.96	-72.03	4000.98	0.5675159676256654
gbmv240_02	123240.66	195652.79	-58.76	3576.21	0.059621963709868164
gbmv240_03	122075.76	188950.65	-54.78	3671.96	0.3770736813695075
gbmv240_04	123883.04	216703.97	-74.93	4413.43	0.04685611586595195
gbmv240_05	121311.73	186714.31	-53.91	4462.87	0.39369572981800705
gbmv480_01	291035.83	487519.59	-67.51	10589.82	0.16327551462745538
gbmv480_02	292048.27	444221.33	-52.10	9201.84	0.7954544343146293
gbmv480_03	293184.37	451659.87	-54.05	10879.43	0.3852361475080668
gbmv480_04	290508.99	469321.17	-61.55	9041.52	0.265618411634559
gbmv480_05	287747.62	437918.84	-52.19	8319.25	0.637818304285135

Tabela 7.2 – Desvio percentual da solução final em relação ao BKV

Instância	Best Known Value (BKV)	Solução Final	Desvio para BKV (%)	Tempo de Execução (segundos)	Semente
gbmv240_01	224964.8	214448.96	4.67	4000.98	0.5675159676256654
gbmv240_02	204624.4	195652.79	4.38	3576.21	0.059621963709868164
gbmv240_03	198937.2	188950.65	5.02	3671.96	0.3770736813695075
gbmv240_04	225683.2	216703.97	3.98	4413.43	0.04685611586595195
gbmv240_05	195521.0	186714.31	4.50	4462.87	0.39369572981800705
gbmv480_01	555993.1	487519.59	12.31	10589.82	0.16327551462745538
gbmv480_02	511107.9	444221.33	13.09	9201.84	0.7954544343146293
gbmv480_03	497652.2	451659.87	9.24	10879.43	0.3852361475080668
gbmv480_04	522604.8	469321.17	10.20	9041.52	0.265618411634559
gbmv480_05	484331.0	437918.84	9.58	8319.25	0.637818304285135

8 CONCLUSÃO

Considera-se que o algoritmo obteve resultados bons para as instâncias do problema com 240 vértices, atingindo desvio percentual máximo de 5.02% em relação ao BKV. Para as instâncias com 480 vértices, o desvio percentual máximo em relação ao BKV aumentou consideravelmente, alcançando 13.09%; apesar disso, considera-se que este resultado também foi satisfatório.

Em comparação com os resultados exatos obtidos por um solver, como o GLPK, fica clara a vantagem do Simulated Annealing: enquanto o solver não conseguiu achar sequer uma solução factível para as instâncias testadas, o Simulated Annealing se aproximou bastante do BKV para todas elas.

Através dos testes realizados para cada parâmetro, propõe-se que se utilize os mesmos valores com os quais os testes das instâncias foram realizados: $r = 0.99$, $I = 2500$, $pf = 0.01$, $pi = 0.95$, $nsi = 10000$ e $k = 2$.

Com base nas considerações acima, conclui-se que o trabalho foi bem-sucedido.

Ok

Avaliação: Um trabalho complete, com todos elementos essencias, e um desenvolvimente excelente da heurística e um trabalho experimental excelente, com uma calibração detalhada de parâmetros, uma análise razoável. A única observação crítica: apresentar médias de replicações torna a análise mais robusta. Uma apresentação boa.
Nota: 10.0.

REFERÊNCIAS

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. **Optimization by Simulated Annealing**. [S.l.]: Science, 1983. 671-680 p. v. 220.

BURIOL, Luciana; RITT, Marcus; COSTA, Alysson M. **INF05010 – Otimização combinatória**: Notas de aula. Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre: [s.n.], 2018. 147-163 p.