TRABALHO DE METAHEURÍSTICAS:

TRAVELING SALESMAN PROBLEM (TSP) WITH DELIVERY

Davi de Paula Oliveira

VIÇOSA

21 de março de 2022

**1. INTRODUÇÃO**

O Problema do Caixeiro Viajante (TSP, *Traveling Salesman Problem*) possui o objetivo de, dado n cidades e a distância entre cada par de cidades, determinar a rota de menor distância total que começa em uma cidade inicial, visita todas as outras cidades exatamente uma vez e volta para a cidade inicial. O TSPd (*Traveling Salesman Problem with delivery*) é uma variante do TSP que possui o mesmo objetivo do TSP, porém o caixeiro viajante pode, ao longo do trajeto, realizar entregas de uma cidade para outra, sendo que ele é remunerado por cada entrega feita. Ou seja, além do conjunto de cidades e da distância entre elas, é dada uma lista de entregas e o valor pago por cada uma delas. Portanto, o custo da rota é proporcional à distância total percorrida, sendo descontados os valores recebidos pelas entregas realizadas. Este relatório possui o objetivo de apresentar três metaheurísticas (GRASP, Iterated Greedy - IG e Ant Colony Optimization - ACO) para o TSPd. A seguir será apresentado as três metaheurísticas detalhadamente, os resultados e as conclusões obtidas.

**2. MÉTODOS**

Antes de detalhar as três metaheurísticas, é importante ressaltar que a cidade inicial sempre será a cidade 1. Além disso, serão considerados dois dados adicionais, k e v, respectivamente o número de entregas e o valor de cada uma (as k entregas sempre serão da cidade 2i para a cidade 2i + 1, com i = 1… k). O caixeiro não é obrigado a fazer todas as entregas, mas só recebe pelas entregas realizadas (isto é, quando a origem da entrega aparece antes do destino na rota do caixeiro). O custo da rota será o valor da distância, descontando as entregas. Por exemplo, uma rota de distância total 1000 com 2 entregas de valor 100 tem valor 1000 − 2 × 100 = 800. Por esse motivo não foi adotada nenhuma estratégia de priorizar as entregas, ou seja, basta tentar reduzir o valor da avaliação da rota, pois a avaliação já desconta os valores das entregas feitas.

As três metaheurísticas possuem como critério de parada 300s, ou seja, após 300s de execução é retornada a melhor solução encontrada após a iteração atual terminar de executar e elas utilizam busca local do tipo best improvement com vizinhança 2-opt. A seguir será descrito detalhadamente o GRASP, o Iterated Greedy e o Ant Colony Optimization.

**2.1 GRASP**

O GRASP foi implementado de forma tradicional, ou seja, são feitas várias iterações (100 no total) e em cada iteração é criado uma solução de forma aleatória-gulosa e, a partir dessa solução, é feita uma busca local. Ao final das 100 iterações ou dos 300s, é retornada a melhor solução encontrada. Além disso, o tamanho da lista restrita de candidatos foi escolhido por cardinalidade, ou seja, max{1, 5% do número de cidades não visitadas}. A seguir é descrito o pseudocódigo:

GRASP(distancias, k, v) {

melhor\_solucao = Nenhum;

avaliacao\_melhor\_solucao = Nenhum;

inicio = tempo atual do sistema;

Para(i = 1 até 100) {

solucao = solucao\_aleatoria\_gulosa(distancias, 0.05);

solucao, avaliacao\_solucao = busca\_local(distancias, k, v, solucao);

Se(i == 1) {

melhor\_solucao = solucao;

avaliacao\_melhor\_solucao = avaliacao\_solucao;

} Ou Se(avaliacao\_solucao < avaliacao\_melhor\_solucao) {

melhor\_solucao = solucao;

avaliacao\_melhor\_solucao = avaliacao\_solucao;

}

fim = tempo atual do sistema;

Se(fim - inicio > 300) {

Saia do Loop;

}

}

retorne melhor\_solucao, avaliacao\_melhor\_solucao;

}

**2.2 Iterated Greedy (IG)**

O Iterated Greedy foi implementado de forma tradicional, ou seja, é criado uma solução inicial de forma gulosa que adiciona sempre a melhor cidade no final da rota e depois é feita uma busca local a partir desta solução. Após isso, são feitas 1000 iterações sendo que cada iteração possui a fase de destruição, construção e busca local. Porém, há duas mudanças principais: a quantidade de elementos que serão removidos e adicionados na solução depende do tamanho dela, ou seja, não é um valor fixo (no caso foi escolhido d = 50% do tamanho da solução); além disso, uma outra modificação é no critério de aceitação, ou seja, a solução atual será trocada apenas se a próxima solução melhorar (no IG original a solução atual também poderá ser trocada mesmo se a próxima solução for pior com uma certa probabilidade). Também é importante ressaltar que na fase de construção, a cidade i que será adicionada na solução é inserida no lugar que menos vai impactar a solução (soma das duas novas arestas menos a aresta antiga). A seguir é descrito o pseudocódigo:

iterated\_greedy(distancias, k, v) {

solucao = solucao\_inicial(distancias);

solucao, avaliacao\_solucao = busca\_local(distancias, k, v, solucao);

melhor\_solucao = solucao;

avaliacao\_melhor\_solucao = avaliacao\_solucao;

inicio = tempo atual do sistema;

Para(i = 1 até 1000) {

solucao\_nova = solucao;

d = 50% do tamanho da solucao\_nova;

Para(i = 1 até d) {

Remova um elemento aleatório da solucao\_nova;

}

Para(i = 1 até d) {

Adicione um elemento removido anteriormente na melhor posiçao da

solucao\_nova;

}

solucao\_nova, avaliacao\_solucao\_nova = busca\_local(distancias, k, v,

solucao\_nova);

Se(avaliacao\_solucao\_nova < avaliacao\_solucao) {

solucao = solucao\_nova;

avaliacao\_solucao = avaliacao\_solucao\_nova;

Se(avaliacao\_solucao < avaliacao\_melhor\_solucao) {

melhor\_solucao = solucao;

avaliacao\_melhor\_solucao = avaliacao\_solucao;

}

}

fim = tempo atual do sistema;

Se(fim - inicio > 300) {

Saia do Loop;

}

}

retorne melhor\_solucao, avaliacao\_melhor\_solucao;

}

**2.3 Ant Colony Optimization (ACO)**

O Ant Colony Optimization foi implementado de forma tradicional, ou seja, primeiro é criado a matriz de feromônios com todas as células com valor 10. Após isso, são criadas k = 30 formigas (soluções) e é escolhida a melhor formiga, ou seja, a rota que possuir o menor valor de avaliação. Uma diferença do algoritmo original é que após identificar a melhor formiga, é feita uma busca local a partir da melhor formiga para melhorar ainda mais a solução. Depois, é feita a atualização da matriz de feromônios, ou seja, é feita a evaporação de todas as células com p = 0.05 e depois é feito a soma de Δ em todas as células da matriz de feromônios que correspondem a cada aresta do caminho feito pela melhor formiga. O processo de criar as formigas e atualizar a matriz de feromônios é feito 1000 vezes. Por fim, é retornado a melhor solução e a avaliação da melhor solução. Além disso, é importante ressaltar que os parâmetros ɑ e β valem 0.2 e 3, respectivamente. A seguir é descrito o pseudocódigo:

ACO(d, k, v) {

inicio = tempo atual do sistema;

melhor\_solucao = Nenhum;

avaliacao\_melhor\_solucao = ∞;

Inicializa a matriz de feromônios t com todas as células com valor 10;

Para(i = 1 até 1000) {

melhor\_formiga = Nenhum;

avaliacao\_melhor\_formiga = ∞;

Para(j = 1 até 30) {

S = [1, 2, 3, …, n];

formiga = [0];

l = 0;

Enquanto(l < n) {

cidade\_escolhida = cidade j com probabilidade ;

formiga.adiciona(cidade\_escolhida);

S.remove(cidade\_escolhida);

l = l + 1;

}

avaliacao\_formiga = avalia\_solucao(formiga, d, k, v);

Se(avaliacao\_formiga < avaliacao\_melhor\_formiga) {

melhor\_formiga = formiga;

avaliacao\_melhor\_formiga = avaliacao\_formiga;

}

}

Realiza a evaporação dos feromônios com p = 0.05;

melhor\_formiga, avaliacao\_melhor\_formiga = busca\_local(d, k, v, melhor\_formiga);

É feito a soma de em todas as células da matriz de feromônios t correspondente a cada aresta do caminho feito pela melhor\_formiga;

Se(avaliacao\_melhor\_formiga < avaliacao\_melhor\_solucao) {

melhor\_solucao = melhor\_formiga;

avaliacao\_melhor\_solucao = avaliacao\_melhor\_formiga;

}

fim = tempo atual do sistema;

Se(fim - inicio > 300) {

Saia do Loop;

}

}

retorne melhor\_solucao, avaliacao\_melhor\_solucao;

}

**3. RESULTADOS**

As metaheurísticas foram implementadas na linguagem Python e testadas em um notebook com processador Intel Core i5-1135G7 (2.4 GHz), 8 GB de RAM e sistema operacional Windows 11. Para testá-las, foram consideradas as seguintes instâncias do TSPLIB, uma biblioteca de instâncias do TSP: gr17, gr21, gr48, gr120, brazil58, berlin52, fri26, eil51, eil76, eil101 e a280. Além disso, apenas as instâncias berlin52, eil51, eil76 e gr48 foram utilizadas para calibrar os parâmetros das metaheurísticas. A seguir é detalhado como foi esse processo.

**3.1 Calibração dos parâmetros do GRASP**

Para realizar a calibração do único parâmetro (α) do GRASP, foram escolhidas cinco possibilidades de valores: 0.05, 0.10, 0.15, 0.20 e 0.25. Para cada uma dessas opções, o GRASP foi executado 5 vezes para cada instância de teste. Após isso, foram criadas 4 tabelas: a tabela 1 mostra as melhores soluções encontradas; a tabela 2 mostra o valor médio das soluções; a tabela 3 mostra o desvio padrão; e a tabela 4 mostra o tempo médio de execução. As tabelas são apresentadas a seguir (os valores em negrito significam que para uma dada instância, o valor em negrito foi o melhor valor encontrado pelo GRASP, além disso, cada instância possui o valor k e v escolhidos):

| **Instância** | **α = 0.05** | **α = 0.10** | **α = 0.15** | **α = 0.20** | **α = 0.25** |
| --- | --- | --- | --- | --- | --- |
| berlin52 3 90 | 7387 | **7272** | 7334 | **7272** | 7326 |
| eil51 0 0 | **429** | 430 | **429** | **429** | 431 |
| eil76 0 0 | **545** | 551 | 551 | 553 | 556 |
| gr48 6 80 | 4672 | **4660** | 4663 | 4663 | 4682 |

*Tabela 1: Melhores Soluções*

| **Instância** | **α = 0.05** | **α = 0.10** | **α = 0.15** | **α = 0.20** | **α = 0.25** |
| --- | --- | --- | --- | --- | --- |
| berlin52 3 90 | 7401,4 | **7360,8** | 7366,2 | 7374 | 7385 |
| eil51 0 0 | **429,6** | 432 | 432,2 | 432,8 | 432,8 |
| eil76 0 0 | **553,2** | 555 | 556,4 | 557,4 | 559 |
| gr48 6 80 | 4677,8 | **4673** | 4680,8 | 4693,4 | 4699 |

*Tabela 2: Valor Médio*

| **Instância** | **α = 0.05** | **α = 0.10** | **α = 0.15** | **α = 0.20** | **α = 0.25** |
| --- | --- | --- | --- | --- | --- |
| berlin52 3 90 | **32,19** | 84,11 | 53,24 | 108,07 | 110,85 |
| eil51 0 0 | **0,89** | 2,12 | 2,58 | 3,27 | 1,3 |
| eil76 0 0 | 5,89 | 2,91 | 4,82 | 3,2 | **2,64** |
| gr48 6 80 | 12,96 | 12,08 | 21,93 | 25,54 | **11,2** |

*Tabela 3: Desvio Padrão*

| **Instância** | **α = 0.05** | **α = 0.10** | **α = 0.15** | **α = 0.20** | **α = 0.25** |
| --- | --- | --- | --- | --- | --- |
| berlin52 3 90 | **15,71** | 19,89 | 28,29 | 33,82 | 32,3 |
| eil51 0 0 | **9,63** | 12,95 | 15,34 | 18,99 | 18,01 |
| eil76 0 0 | **46,78** | 72,93 | 81,88 | 109,41 | 105,34 |
| gr48 6 80 | **14,67** | 20,81 | 22,17 | 29,68 | 29,37 |

*Tabela 4: Tempo Médio (s)*

Pelas tabelas acima é possível concluir que: os α’s que obtiveram as melhores soluções foram 0.05, 0.10 e 0.20; os α’s que obtiveram os melhores valores médio foram 0.05 e 0.10; os α’s que obtiveram os melhores desvios padrão foram 0.05 e 0.25; e o α que obteve o melhor tempo médio foi o 0.05. Portanto, o melhor α é o 0.05 pois com ele foi obtido as melhores soluções, os melhores valores médio, os melhores desvios padrão e o melhor tempo médio.

**3.2 Calibração dos parâmetros do Iterated Greedy (IG)**

Para realizar a calibração do único parâmetro (d) do Iterated Greedy, foram escolhidas cinco possibilidades de valores: 0.10, 0.25, 0.50, 0.75 e 0.90. Para cada uma dessas opções, o Iterated Greedy também foi executado 5 vezes para cada instância de teste. A seguir são apresentadas as 4 tabelas iguais às criadas para o GRASP:

| **Instância** | **d = 0.10** | **d = 0.25** | **d = 0.50** | **d = 0.75** | **d = 0.90** |
| --- | --- | --- | --- | --- | --- |
| berlin52 3 90 | 7535 | 7535 | **7272** | **7272** | **7272** |
| eil51 0 0 | 429 | **427** | **427** | **427** | **427** |
| eil76 0 0 | 544 | **538** | **538** | **538** | 551 |
| gr48 6 80 | 4835 | **4660** | **4660** | **4660** | **4660** |

*Tabela 5: Melhores Soluções*

| **Instância** | **d = 0.10** | **d = 0.25** | **d = 0.50** | **d = 0.75** | **d = 0.90** |
| --- | --- | --- | --- | --- | --- |
| berlin52 3 90 | 7541,8 | 7535 | **7272** | **7272** | **7272** |
| eil51 0 0 | 429 | 427,8 | **427** | **427** | 427,4 |
| eil76 0 0 | 544 | 541,6 | **540,4** | 540,6 | 551,2 |
| gr48 6 80 | 4835 | 4661,8 | **4660** | **4660** | 4660,2 |

*Tabela 6: Valor Médio*

| **Instância** | **d = 0.10** | **d = 0.25** | **d = 0.50** | **d = 0.75** | **d = 0.90** |
| --- | --- | --- | --- | --- | --- |
| berlin52 3 90 | 15,2 | **0** | **0** | **0** | **0** |
| eil51 0 0 | **0** | 0,44 | **0** | **0** | 0,54 |
| eil76 0 0 | **0** | 3,28 | 3,28 | 2,79 | 0,44 |
| gr48 6 80 | **0** | 1,64 | **0** | **0** | 0,44 |

*Tabela 7: Desvio Padrão*

| **Instância** | **d = 0.10** | **d = 0.25** | **d = 0.50** | **d = 0.75** | **d = 0.90** |
| --- | --- | --- | --- | --- | --- |
| berlin52 3 90 | **9,8** | 13,61 | 25,2 | 47 | 62,29 |
| eil51 0 0 | 8,83 | **8,43** | 15,35 | 17,43 | 22,36 |
| eil76 0 0 | **26,55** | 30,7 | 63,8 | 94,35 | 103,48 |
| gr48 6 80 | **10,26** | 14,21 | 26,44 | 35,76 | 46,26 |

*Tabela 8: Tempo Médio (s)*

Pelas tabelas acima é possível concluir que: os d’s que obtiveram as melhores soluções foram 0.50 e 0.75; o d que obteve o melhor valor médio foi 0.50; os d’s que obtiveram os melhores desvios padrão foram 0.10, 0.50 e 0.75; e o d que obteve o melhor tempo médio foi o 0.10. Portanto, o melhor d é o 0.50 pois com ele foi obtido as melhores soluções, os melhores valores médio e os melhores desvios padrão.

**3.3 Calibração dos parâmetros do Ant Colony Optimization (ACO)**

Para realizar a calibração dos parâmetros do Ant Colony Optimization, foram escolhidas cinco possibilidades de conjunto de valores para ɑ, β, p e k: 1 - (0.1, 2, 0.1, 10), 2 - (2, 0.1, 0.01, 50), 3 - (1, 1, 0.2, 25), 4 - (1, 2, 0.15, 20) e 5 - (0.2, 3, 0.05, 30). Para cada uma dessas opções, o Ant Colony Optimization foi executado 5 vezes para cada instância de teste. A seguir são apresentadas as 4 tabelas iguais às criadas para o IG:

| **Instância** | **1** | **2** | **3** | **4** | **5** |
| --- | --- | --- | --- | --- | --- |
| berlin52 3 90 | **7272** | **7272** | 7326 | **7272** | **7272** |
| eil51 0 0 | **426** | 427 | 429 | 428 | **426** |
| eil76 0 0 | 548 | 548 | 544 | **538** | 544 |
| gr48 6 80 | **4660** | **4660** | 4662 | 4663 | **4660** |

*Tabela 9: Melhores Soluções*

| **Instância** | **1** | **2** | **3** | **4** | **5** |
| --- | --- | --- | --- | --- | --- |
| berlin52 3 90 | **7272** | 7325 | 7375 | 7325 | **7272** |
| eil51 0 0 | 428,8 | 428,2 | 431,2 | 429,4 | **427,2** |
| eil76 0 0 | 552 | 554 | 547,2 | **541,6** | 547,4 |
| gr48 6 80 | 4662,8 | **4661,4** | 4670 | 4668,4 | 4661,8 |

*Tabela 10: Valor Médio*

| **Instância** | **1** | **2** | **3** | **4** | **5** |
| --- | --- | --- | --- | --- | --- |
| berlin52 3 90 | **0** | 32,38 | 37,2 | 52,35 | **0** |
| eil51 0 0 | 1,64 | 1,3 | 1,78 | 1,34 | **1,09** |
| eil76 0 0 | 3,8 | 5,04 | **2,16** | 2,3 | 3,2 |
| gr48 6 80 | 3,03 | 1,51 | 16,23 | 5,07 | **1,3** |

*Tabela 11: Desvio Padrão*

| **Instância** | **1** | **2** | **3** | **4** | **5** |
| --- | --- | --- | --- | --- | --- |
| berlin52 3 90 | 300,19 | 300,65 | 139,18 | **122,02** | 300,17 |
| eil51 0 0 | 149,86 | 300,13 | 76,99 | **66,01** | 162,67 |
| eil76 0 0 | 300,31 | 301,01 | 214,83 | **187,16** | 300,35 |
| gr48 6 80 | 300,23 | 300,54 | 141,14 | **127,53** | 300,2 |

*Tabela 12: Tempo Médio (s)*

Pelas tabelas acima é possível concluir que: os conjuntos de valores que obtiveram as melhores soluções foram os conjuntos 1 e 5; o conjunto de valores que obteve o melhor valor médio foi o conjunto 5; o conjunto de valores que obteve o melhor desvio padrão foi o conjunto 5; e o conjunto de valores que obteve o melhor tempo médio foi o conjunto 4. Portanto, o melhor conjunto de valores para os parâmetros é o conjunto 5, ou seja, ɑ = 0.2, β = 3, p = 0.05 e k = 30, pois com eles foram obtidos as melhores soluções, os melhores valores médio e os melhores desvios padrão.

**3.4 Comparação dos resultados**

Após escolher α = 0.05 para o GRASP, d = 0.50 para o IG e (ɑ, β, p, k) = (0.2, 3, 0.05, 30) para o ACO, todas as instâncias foram executadas 5 vezes e foram extraídas o valor da solução e o tempo de execução. A seguir é apresentado uma tabela que compara os valores obtidos pelo GRASP, IG e ACO (os valores em negrito indicam qual foi o melhor valor encontrado para uma dada instância):

| **Instância** | **Metaheurística** | **Melhor Solução** | **Valor Médio** | **Desvio Padrão** | **Tempo Médio (s)** |
| --- | --- | --- | --- | --- | --- |
| gr17 0 0 | GRASP | **2085** | **2085** | **0** | **0,12** |
|  | IG | **2085** | **2085** | **0** | 0,6 |
|  | ACO | **2085** | **2085** | **0** | 13,21 |
| gr21 0 0 | GRASP | 2795 | 2795 | **0** | **0,43** |
|  | IG | **2707** | **2707** | **0** | 0,65 |
|  | ACO | **2707** | **2707** | **0** | 19,46 |
| gr48 0 0 | GRASP | 5066 | 5066 | **0** | **8,81** |
|  | IG | **5046** | **5046** | **0** | 14,36 |
|  | ACO | **5046** | **5046** | **0** | 152,07 |
| gr120 0 0 | GRASP | 7061 | 7090 | **36,49** | **301,96** |
|  | IG | **6956** | **6978,4** | 36,88 | 302,62 |
|  | ACO | 7070 | 7177,6 | 65,28 | 302,23 |
| brazil58 0 0 | GRASP | **25395** | **25395** | **0** | **23,07** |
|  | IG | **25395** | **25395** | **0** | 47,21 |
|  | ACO | **25395** | **25395** | **0** | 287,05 |
| berlin52 0 0 | GRASP | 7657 | 7657 | **0** | **11,76** |
|  | IG | **7542** | **7542** | **0** | 18,4 |
|  | ACO | **7542** | **7542** | **0** | 178,95 |
| fri26 0 0 | GRASP | 955 | 955 | **0** | **0,54** |
|  | IG | **937** | **937** | **0** | 1,88 |
|  | ACO | **937** | **937** | **0** | 25,99 |
| eil51 0 0 | GRASP | 429 | 429,6 | 0,89 | **9,63** |
|  | IG | 427 | **427** | **0** | 15,35 |
|  | ACO | **426** | 427,2 | 1,09 | 162,67 |

| eil76 0 0 | GRASP | 545 | 553,2 | 5,89 | **46,78** |
| --- | --- | --- | --- | --- | --- |
|  | IG | **538** | **540,4** | 3,28 | 63,8 |
|  | ACO | 544 | 547,4 | **3,2** | 300,35 |
| eil101 0 0 | GRASP | 641 | 651,8 | 6,14 | 166,27 |
|  | IG | **629** | **629,4** | **0,89** | **158,59** |
|  | ACO | 635 | 647,6 | 10,18 | 302,17 |
| a280 0 0 | GRASP | 2751 | 2878,6 | 73,3 | 367,96 |
|  | IG | **2663** | **2715,4** | **29,98** | 337,34 |
|  | ACO | 2770 | 2810 | 46,18 | **329,81** |
| gr17 3 80 | GRASP | **1848** | **1848** | **0** | **0,17** |
|  | IG | **1848** | **1848** | **0** | 0,69 |
|  | ACO | **1848** | **1848** | **0** | 13,9 |
| gr48 6 80 | GRASP | 4672 | 4677,8 | 12,96 | **14,67** |
|  | IG | **4660** | **4660** | **0** | 26,44 |
|  | ACO | **4660** | 4661,8 | 1,3 | 300,2 |
| gr48 10 80 | GRASP | 4561 | 4561 | **0** | **17,8** |
|  | IG | **4511** | **4511** | **0** | 38,1 |
|  | ACO | 4512 | 4529,6 | 19,88 | 300,2 |
| brazil58 5 300 | GRASP | **24195** | **24195** | **0** | **33,04** |
|  | IG | **24195** | **24195** | **0** | 76,89 |
|  | ACO | **24195** | **24195** | **0** | 300,47 |
| brazil58 8 100 | GRASP | **24845** | **24845** | **0** | **46,64** |
|  | IG | **24845** | **24845** | **0** | 94,94 |
|  | ACO | **24845** | **24845** | **0** | 300,72 |
| brazil58 8 250 | GRASP | 24105 | 24105 | **0** | **40,21** |
|  | IG | **23945** | **23945** | **0** | 82,48 |
|  | ACO | **23945** | **23945** | **0** | 300,72 |
| berlin52 3 90 | GRASP | 7387 | 7401,4 | 32,19 | **15,71** |
|  | IG | **7272** | **7272** | **0** | 25,2 |
|  | ACO | **7272** | **7272** | **0** | 300,17 |
| eil101 10 5 | GRASP | 620 | 624,4 | 6,38 | **301,67** |
|  | IG | **596** | **596,8** | **1,78** | 302,59 |
|  | ACO | 626 | 633,8 | 7,42 | 311,46 |

| a280 10 10 | GRASP | 2792 | 3056,8 | 150,33 | 423,22 |
| --- | --- | --- | --- | --- | --- |
|  | IG | **2660** | **2673** | **11,61** | **360,76** |
|  | ACO | 2883 | 3033,8 | 113,99 | 963,59 |

*Tabela 13: GRASP x IG x ACO*

Apesar do GRASP e IG serem parecidos, a partir dos dados acima, é possível concluir que em relação à qualidade da solução (melhor solução, valor médio e desvio padrão), o IG teve um desempenho melhor. Porém, em relação ao tempo de execução, o GRASP obteve vantagem. Uma possibilidade do IG ser melhor na qualidade da solução e pior em relação ao tempo de execução é que ele diversifica mais o espaço de busca pois o valor de d é maior do que o de α. Além disso, uma outra possibilidade seria em relação ao número de iterações pois enquanto o GRASP executa no máximo 100 iterações, o IG executa no máximo 1000. Esses valores foram escolhidos pois ao longo dos testes foi observado que o GRASP executa menos iterações por minuto se compararmos com o IG. Ou seja, quando o GRASP era executado com 1000 iterações, praticamente todas as instâncias executavam com um tempo de 300s (tempo limite). Portanto, para reduzir o tempo de testes, foi escolhido um número de iterações menor para que apenas as instâncias grandes demorem 300s.

E comparando o ACO com o GRASP e o IG, é possível concluir que em relação à qualidade da solução (melhor solução, valor médio e desvio padrão), o ACO é melhor que o GRASP mas é pior que o IG. No entanto, em relação ao tempo de execução, o ACO é a heurística mais lenta justamente por ser uma heurística populacional, ou seja, uma heurística que trabalha com várias soluções ao mesmo tempo. Uma das possibilidades do ACO ser melhor que o GRASP em relação à qualidade da solução é que, por ser uma heurística populacional, o ACO consegue diversificar melhor a solução para intensificar com uma busca local. E o ACO é pior que o IG em relação a qualidade da solução por causa do pouco tempo para executar as instâncias (principalmente as maiores), pois o ACO é mais lento que o IG, ou seja, o ACO não consegue diversificar e intensificar a solução de forma mais eficiente que o IG.

Uma outra comparação que podemos fazer é em relação aos melhores valores obtidos em todas as etapas do trabalho: parte 0 (solução crescente), parte 1.1 (solução gulosa que sempre adiciona no final da rota), parte 1.2 (solução gulosa que pode adicionar no início ou no fim da rota), parte 2.1 (busca local a partir da solução da parte 1.1), parte 2.2 (busca local a partir da solução da parte 1.2), GRASP, IG e ACO. A seguir é apresentado uma tabela comparando esses valores (os valores em negrito indicam a melhor solução encontrada para uma dada instância):

| **Instância** | **Parte 0** | **Parte 1.1** | **Parte 1.2** | **Parte 2.1** | **Parte 2.2** | **GRASP** | **IG** | **ACO** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| gr17 0 0 | 4722 | 2187 | 2187 | **2085** | **2085** | **2085** | **2085** | **2085** |
| gr21 0 0 | 6620 | 3333 | 3338 | 2795 | 2782 | 2795 | **2707** | **2707** |
| gr48 0 0 | 19837 | 6098 | 6148 | 5259 | 5136 | 5066 | **5046** | **5046** |
| gr120 0 0 | 50021 | 9351 | 9277 | 7231 | 7214 | 7061 | **6956** | 7070 |
| brazil58 0 0 | 129267 | 30774 | 29194 | 26008 | 26008 | **25395** | **25395** | **25395** |
| berlin52 0 0 | 22205 | 8980 | 8790 | 7842 | 8056 | 7657 | **7542** | **7542** |
| fri26 0 0 | 1140 | 1112 | 1090 | 955 | 953 | 955 | **937** | **937** |
| eil51 0 0 | 1308 | 511 | 555 | 435 | 437 | 429 | 427 | **426** |
| eil76 0 0 | 1969 | 642 | 673 | 562 | 563 | 545 | **538** | 544 |
| eil101 0 0 | 2062 | 803 | 855 | 637 | 640 | 641 | **629** | 635 |
| a280 0 0 | 2808 | 3157 | 3182 | 2756 | 2734 | 2751 | **2663** | 2770 |
| gr17 3 80 | 4482 | 2107 | 2027 | **1848** | 1925 | **1848** | **1848** | **1848** |
| gr48 6 80 | 19357 | 5778 | 5908 | 4967 | 4816 | 4672 | **4660** | **4660** |
| gr48 10 80 | 19037 | 5538 | 5748 | 4854 | 4656 | 4561 | **4511** | 4512 |
| brazil58 5 300 | 127767 | 29574 | 27994 | 24808 | 24808 | **24195** | **24195** | **24195** |
| brazil58 8 100 | 128467 | 30274 | 28594 | 25445 | 25445 | **24845** | **24845** | **24845** |
| brazil58 8 250 | 127267 | 29524 | 27694 | 24887 | 24887 | 24105 | **23945** | **23945** |
| berlin52 3 90 | 21935 | 8890 | 8610 | 7662 | 7876 | 7387 | **7272** | **7272** |
| eil101 10 5 | 2012 | 793 | 845 | 630 | 620 | 620 | **596** | 626 |
| a280 10 10 | 2708 | 3087 | 3152 | 2702 | 2667 | 2.792 | **2.660** | 2883 |

*Tabela 14: Valores obtidos em todas as etapas*

Pelos dados da tabela acima é possível concluir novamente que o IG foi o algoritmo mais eficiente para encontrar uma solução, seguido pelo ACO e depois pelo GRASP. Além disso, é possível observar que o GRASP nem sempre é melhor que as partes 2.1 e 2.2. Isso ocorre nas instâncias: gr21 0 0, fri26 0 0, eil101 0 0, a280 0 0 e a280 10 10. Os motivos do GRASP nem sempre melhorar podem ser devido ao baixo número de iterações (100) e ao limite máximo de execução (300s) ser bem baixo. E o ACO também nem sempre é melhor que as partes 2.1 e 2.2. Isso ocorre nas instâncias: a280 0 0, eil101 10 5 e a280 10 10. O principal motivo do ACO nem sempre melhorar é devido ao limite máximo de execução (300s) ser bem baixo.

**4. CONCLUSÃO**

Foram apresentadas três metaheurísticas (GRASP, Iterated Greedy - IG e Ant Colony Optimization - ACO) para resolver o problema do caixeiro viajante com entregas. Conforme pode ser observado a partir das tabelas 13 e 14, o IG conseguiu produzir as melhores soluções finais em praticamente todos os casos. O IG também mostrou ser mais robusto por gerar soluções finais com menor desvio em relação às melhores soluções encontradas para cada instância. Porém é importante ressaltar que não é possível afirmar que esses fatos comprovam a supremacia do IG sobre o GRASP e o ACO pois foram utilizadas poucas instâncias para a calibragem dos parâmetros, o número de iterações do GRASP foi baixo devido ao tempo de execução e o tempo limite de execução das três metaheurísticas também foi baixo.