TÉCNICO LISBOA

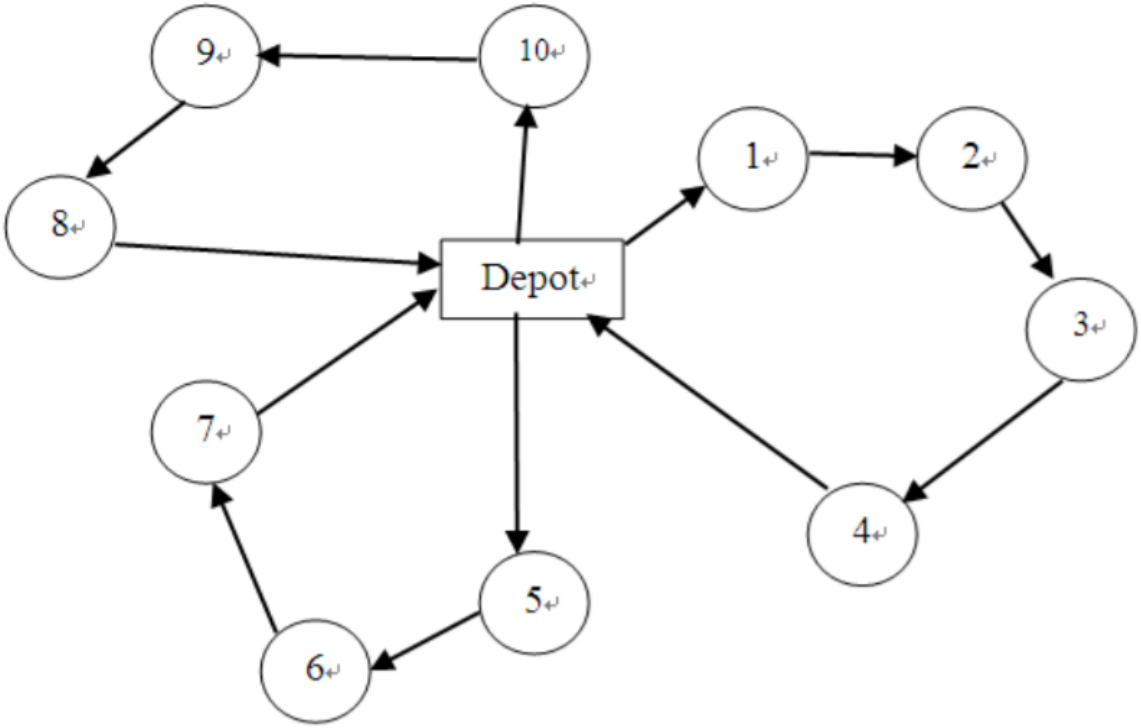
Capacitated Vehicle Routing Problem

Optimização e Decisão

Instituto Superior Técnico

MEMec

Julho 2021



Catarina Pires, Nº 90230

Professores: João Sousa, Miguel Martins

Índice

1 Introdução

Neste projeto no âmbito da unidade curricular de Otimização e Decisão será resolvido um problema de *Capacitated Vehicle Routing Problem* recorrendo a duas metaheurísticas, Otimização por Colónia de Formigas (*Ant Colony Optimization* (ACO)) e Algoritmo Genético (*Genetic Algorithm* (GA)). Neste relatório será apresentada uma introdução ao problema de CVRP e descrição do problema, serão apresentados os métodos utilizados e a sua implementação e, finalmente, serão analisados os resultados obtidos para cada um dos métodos, para três instâncias do problema e comparados entre si e com as soluções ótimas para cada instância.

Como referido por Chen, Hsieh e Wu [4], recentemente, o ACO tem sido cada vez mais usado para resolver problemas onde previamente se utilizava o GA, dadas as semelhanças existentes entre estes dois algoritmos ("*Recently, ACO has been widely used in the areas where genetic algorithm was used previously since some similarities exist between algorithms.*"). Desta forma, achou-se interessante a comparação do desempenho destes dois métodos quando aplicados ao problema CVRP.

Contrastando com o problema de CVRP previamente abordado, neste relatório apenas é tratado um problema CVRP genérico, utilizando como conjunto de dados três instâncias de Christofides e Eilon [16]. Para além da dimensão das instâncias analisadas ser maior, este relatório também se diferencia pela utilização de uma nova metaheurítica, ACO, e por utilizar um GA feito de raiz, sendo que anteriormente se recorreu à *toolbox* de otimização do *MATLAB*.

2 Descrição do Problema

O problema *Vehicle Routing Problem* (VRP) é um conhecido problema combinatório de otimização, o qual tem um papel essencial em logística e gestão de cadeias de distribuição devido às suas variadas aplicações em transportação, entrega de produtos, e serviços. Como referido em [7], os custos de transportação são uma componente muito significativa do custo total de um produto (10% a 15% em média), salientando assim a importância de encontrar as melhores soluções para o problema VRP. Este problema é uma generalização do problema *Travelling Salesman Problem*, tratando-se então de um problema *NP-hard*, o que significa que apenas é possível obter soluções ótimas, utilizando métodos exatos, para instâncias de pequena dimensão. Para instâncias de maior dimensão recorre-se a métodos heurísticos para encontrar boas soluções num tempo de computação razoável, não garantindo resultados ótimos.

O problema *Capacitated Vehicle Routing Problem* (CVRP), inicialmente proposto por Dantzig e Ramser [8] em 1959, é uma variante do problema VRP, sendo constituído por um único centro de distribuição, uma frota de veículos semelhantes entre si, e um conjunto de clientes. Cada veículo possui uma capacidade fixa da quantidade máxima de produtos que pode transportar, e o objetivo do problema é encontrar o conjunto de rotas que permite a satisfação dos pedidos de todos os clientes, com a mínima distância percorrida possível, consequentemente minimizando o custo.

2.1 Formulação do Problema

Dado um número n de clientes que devem ser servidos, definindo N como o conjunto de todos os clientes tal que $N\{1,2,...,n\}$, V é o número de nós (número de clientes + centro de distribuição), tal que $V = \{0\} \cup N$, e A é o conjunto de arcos possíveis $A = \{(i,j) \in v^2 : i \neq j\}$. A matriz de distâncias associadas a cada arco $(i,j) \in A$ é dada por d_{ij}, a quantidade de produtos que deve ser

entregue a cada cliente $i \in N$ é dada por q_i, e quantidade de produtos máxima que um veículo pode transportar é dada por Q. Finalmente, define-se $x_{i,j}$ como,

$$x_{i,j} = \begin{cases} 1 & \text{se um veículo vai do cliente i para o cliente j} \\ 0 & \text{Caso contrário} \end{cases} \qquad i,j \in A \tag{1}$$

Uma vez definidos os parâmetros e variáveis do problema, este pode ser formulado da seguinte forma:

Pretende-se minimizar a distância das viagens, portanto a função objetivo é dada por:

$$\min \quad \sum_{i,j \in A} d_{ij} x_{ij} \tag{2}$$

O problema está sujeito aos seguintes constrangimentos:

- Cada cliente é visitado por apenas um veículo:

$$\sum_{j \in V, j \neq i} x_{ij} = 1 \quad i \in N \tag{3}$$

- Apenas um veículo parte de cada cliente (o mesmo veículo que o visitou):

$$\sum_{i \in V, i \neq j} x_{ij} = 1 \quad j \in N \tag{4}$$

- A capacidade dos pedidos satisfeitos u_i não pode ser maior que a capacidade do veículo:

$$\text{se } x_{ij} = 1 \;\Rightarrow\; u_i + q_j = u_j \quad i,j \in A : j \neq 0, i \neq 0 \tag{5}$$

$$q_i \leq u_i \leq Q \quad i \in N \tag{6}$$

2.2 Dados Utilizados

Os conjuntos de dados utilizados na resolução deste trabalho tratam-se de três instâncias de Christofides e Eilon [16] para o problema de CVRP:

- **E-n22-k4:** Devem ser servidos 21 clientes e a capacidade máxima de cada veículo é 6000 produtos. A solução ótima é dada por 4 rotas e a distância percorrida é 375 km;
- **E-n51-k5:** Devem ser servidos 50 clientes e a capacidade máxima de cada veículo é 160 produtos. A solução ótima é dada por 5 rotas e a distância percorrida é 521 km;
- **E-n101-k14:** Devem ser servidos 100 clientes e a capacidade máxima de cada veículo é 112 produtos. Não foi encontrada uma solução ótima para esta instância. Segundo o *website* de onde se obtiveram os conjuntos de dados [16] a melhor solução consta de 14 rotas e uma distância total percorrida de 1077 km. Por outro lado, segundo o *website* The VRP Web [18] na melhor solução conhecida a distância percorrida é igual a 1071 km, não fornecendo o número total de rotas. Uma vez que o resultado apresentado em [18] também se verificou em [19] e [20], optou-se por utilizar este valor como referência;

As distâncias entre nós são distâncias euclidianas.

3 Otimização por Colónia de Formigas

A Otimização por Colónia de Formigas (*Ant Colony Optimization* (ACO)) foi apresentada por Dorigo e Stutzle (2004) [2], e esta é inspirada pelo comportamento de formigas quando estas estão à procura de alimento. Neste processo natural as formigas libertam uma substância química, as feromonas, indicando o caminho que estão a percorrer. O trilho de feromonas é uma forma de comunicação entre as formigas que atrairá outras formigas para percorrer o mesmo caminho. Desta forma, quanto maior a concentração de feromonas num certo caminho, maior é a probabilidade da próxima formiga escolher percorrer esse caminho. Ao longo do tempo, à medida que mais formigas completam o caminho mais curto, há uma acumulação de feromonas mais rápida no percurso mais curto do que no percurso mais longo.

As formigas artificiais que constituem o ACO movem-se em grafos, isto é, apenas pelos nós e arcos existentes, e o método é discreto, tornando-o inerentemente um bom método para resolver problemas combinatórios. Baseando-se no comportamento descrito acima, as formigas artificiais também escolhem um caminho de acordo com a quantidade de feromonas, depositando feromonas pelos caminhos por onde passam, sendo que a cada iteração o algoritmo atualiza a quantidade de feromonas. As formingas artificiais conseguem ainda "ver", ou seja, também utilizam uma heurística para a decisão do caminho, e possuem memória mantendo e atualizando um conjunto de nós potenciais N.

3.1 Otimização por Colónia de Formigas Aplicado ao CVRP

As etapas principais consideradas num ACO são a construção da solução, a gestão dos trilhos de feromonas, e métodos adicionais, como heurística [3].

3.1.1 Construção da Solução

No ACO cada formiga artificial simula um veículo, e o seu conjunto completo de rotas é construído pela sucessiva escolha do próximo cliente a visitar, até que todos os clientes tenham sido visitados. Uma nova rota será começada a partir do centro de distribuição quando a escolha do próximo cliente leva a um solução impossível devido à capacidade máxima do veículo ser ultrapassada. Consequentemente, há um total de m soluções construídas sequencialmente pelo total de m formigas artificiais em cada iteração.

Inicialmente cada formiga é atribuída a um cliente escolhido aleatoriamente para visitar assim que sai do centro de distribuição. De seguida, a cada etapa da construção, uma formiga k, num certo cliente i, escolherá o próximo cliente j a ser visitado a partir de um conjunto de vizinhança possível N de acordo com a probabilidade,

$$p_{ij}^{k} = \begin{cases} \frac{\tau_{ij}^{\alpha} \times \eta_{ij}^{\beta}}{\sum_{j \in N}} & \text{se } j \in N \\ 0 \quad \text{Caso contrário} \end{cases} \tag{7}$$

Onde $\eta_{ij} = 1/d_{ij}$ é a heurística e τ_{ij} denota a concentração de feromonas no caminho que liga o cliente i ao cliente j. Os parâmetros α e β regulam a influencia relativa da concentração de feromonas e da heurística na decisão do próximo nó. Segundo a equação 7 a seleção de um cliente que ainda não foi visitado depende dos seguintes critérios:

- Concentração de feromonas, τ_{ij} o qual indica quão boa é a escolha do próximo cliente j partindo da atual cidade i;

- Atratividade, η_{ij} o qual indica quão promissora é a escolha do próximo cliente j partindo da atual cidade i;
- Conjunto de vizinhança possível, N, o qual também se pode chamar lista de candidatos, que inclui os clientes mais próximos do cliente atual i que podem ser selecionados como o próximo nó da rota.

A probabilidade de escolher um certo arco (i,j) aumentará com o aumento da respetiva concentração de feromonas τ_{ij}, enquanto que o valor da heurística não irá mudar dinamicamente ao longo do tempo. Quando todos os clientes da lista de candidatos tiverem sido visitados pelas formigas será escolhido um cliente que não pertence à lista de candidatos. Neste caso, uma formiga k selecionará o cliente com o maior valor de $\tau_{ij}^{\alpha}\eta_{ij}^{\beta}$. A utilização de uma lista de candidatos permite uma redução significativa do tempo de computação necessário para as formigas formarem soluções, dado que têm de escolher a partir de um menor grupo de clientes. Durante a construção das rotas, uma formiga k regressa ao centro de distribuição quando o constrangimento da capacidade máxima do veículo é violado. Seguidamente, a mesma formiga k, que representa um veículo, começará uma nova rota para servir os clientes que ainda não foram visitados. Este processo repete-se até todos os clientes serem visitados.

3.1.2 Heurística

Depois de uma formiga artificial terminar de construir a sua solução, mas antes da próxima formiga começar a cosntruir a sua solução, há uma atualização das feromonas e a solução da formiga anterior é melhorada aplicando a heurística.

3.1.3 Atualização das feromonas

Após todas as soluções das formigas artificiais terem sido melhoradas recorrendo à heurística, as concentrações de feromonas nos caminhos são atualizadas. Esta é a característica principal do ACO que permite melhorar soluções futuras, uma vez que os trilhos de feromonas atualizados refletem o desempenho das formigas e a qualidade das suas soluções. A atualização das feromonas é constituída por duas fases, a deposição e a evaporação de feromonas. No algoritmo utilizado foram feitas algumas modificações ao processo habitual de evaporação de feromonas, sendo que este é feito de acordo com Bullnheimer *et al.* (1999) [9], o qual se serve de uma estratégia elitista e do conceito de *ranking*.

- **Evaporação de Feromonas:** Primeiramente, a concentração de feromonas em todos os arcos será reduzida através de um fator constante,

$$\tau_{ij} = (\rho + \frac{\theta}{L^{avg}})\tau_{ij}, \forall (i,j) \in A \tag{8}$$

Onde $0 \leq (\rho + \frac{\theta}{L^{avg}}) < 1$ é o fator de evaporação, $0 \leq \rho < 1$ é a percentagem do trilho de feromonas que se mantém, θ é uma constante, e $L^{avg} = \sum_{k=1}^{m} \frac{L^k}{m}$ é a média da distância total encontrada por m formigas artificiais por iteração, e L^k é a distância total obtida pela solução de uma formiga artificial k. Este ACO utiliza $(\rho + \frac{\theta}{L^{avg}})$ como fator de evaporação, em vez de apenas ρ como formulado por Dorigo e Stutzle (2004) [2]. A utilização deste factor visa simular o processo de evaporação de feromonas na natureza, o qual depende do comprimento do caminho percorrido pela formiga. Quanto maior é o caminho, mais feromonas irá evaporar, incentivando a exploração de arcos ainda não visitados, tornando os arcos já visitados menos atrativos. Além disso, este processo evita a convergência rápida e precoce das formigas para uma solução subótima.

- **Depositação de Feromonas:** Após o processo de evaporação de feromonas, apenas as melhores formigas e as formigas elitistas depositarão feromonas nos arcos por onde passaram, de acordo com,

$$\tau_{ij} = \tau_{ij} + \sum_{\lambda=1}^{\sigma-1} \Delta\tau_{ij}^{\lambda} + \Delta\tau_{ij}* \tag{9}$$

Onde

$$\Delta\tau_{ij}^{\lambda} = \begin{cases} \frac{\sigma-\lambda}{L^{\lambda}} & \text{se a } \lambda\text{-ésima melhor formiga viaja no arco (i,j)} \\ 0 & \text{Caso contrário} \end{cases} \tag{10a}$$

$$\Delta\tau_{ij}* = \begin{cases} \frac{\sigma}{L*} & \text{se o arco (i,j) é parte da melhor solução} \\ 0 & \text{Caso contrário} \end{cases} \tag{10b}$$

Desta forma, dois tipos de trilhos de feromonas são depositados durante a atualização das feromonas. Primeiramente, a melhor solução encontrada até ao momento desde o inicio do algoritmo será atualizada como se σ formigas elitistas o tivessem atravessado. A quantidade de feromonas depositada pelas formigas elitistas é dada por $\Delta\tau_{ij}*$. De seguida, apenas as $\sigma - 1$ melhores formigas, das m formigas na atual iteração, podem depositar feromonas nos caminhos por onde passaram. A quantidade de feromonas depositada por estas formigas depende da sua classificação (*rank*) λ e também da qualidade da sua solução L^k, em que a λ-ésima melhor formiga deposita a quantidade de feromonas dada por $\Delta\tau_{ij}^{\lambda}$. A estratégia elitista pretende valorizar os arcos pertencentes às melhores soluções encontradas em cada iteração, visto que é provável que alguns destes pertençam à solução ótima. Por outro lado, o conceito de classificação visa evitar a sobrestimação de trilhos de feromonas provenientes de formigas que percorreram rotas subotimas.

Por fim, o pseudocódigo deste algoritmo pode ser escrito da seguinte forma,

Algorithm 1 Ant Colony Optimization

```
1:  Inicialização
2:  Definir τ_ij = τ_0
3:  for l = 1 to N_max do
4:      Construir o caminho
5:      for i = 1 to n do
6:          for k = 1 to m do
7:              Escolher nó
8:              Atualizar N
9:              Aplicar heurística local
10:     Analisar a solução
11:     for k = 1 to m do
12:         Calcular f_k
13:     Atualizar as feromonas
```

3.2 Implementação Computacional

A implementação do ACO foi obtida em [17], à qual se realizaram as seguintes alterações: adicionou-se o contador de tempo de computação, e definiu-se τ_0 como uma variável de forma a possibilitar que o utilizador mude este valor mais facilmente. A implementação foi feita na linguagem de programação *python*, encontrando-se no ficheiro *ACO_CVRP.py*, e para correr este ficheiro utilizou-se o software *PyCharm Community Edition 2020.3.3*.

Uma vez aberto o ficheiro, o utilizador deverá escrever o nome do ficheiro *.txt* que contem os dados da instância que pretende correr, e definir os valores de α (variável $alfa$), β (variável *beta*), σ (variável *sigm*), ρ (variável *ro*), θ (variável *th*), τ_0 (variável *tauZero*), número máximo de iterações (variável *iterations*), e número de formigas artificiais (variável *ants*). De seguida, o utilizador poderá correr o código.

O ficheiro começa por criar a variável *bestSolution* e, a partir do ficheiro *.txt*, definir o conjunto de vértices (nós) do problema, a matriz de distâncias entre nós, a capacidade máxima do veículo, a matriz de pedidos de cada cliente, a matriz de feromonas, e guarda também o valor da melhor solução encontrada para a respetiva instância. De seguida, para cada iteração, obtém uma solução para cada formiga artificial e calcula a respetiva distância total percorrida, atualiza a matriz de feromonas, e escreve na janela de comandos o melhor valor obtido na respetiva iteração.

As funções criadas para a implementação do ACO foram as seguintes:

- ***generateGraph*:** Esta função cria o grafo que contém todos os clientes e o centro de distribuição. Recorrendo ao ficheiro *RegExService.py*, esta função lê o ficheiro *.txt* definindo os valores da capacidade máxima de cada veículo, as coordenadas de cada nó, a matriz de procura de cada cliente, e o valor da melhor solução encontrada para a respetiva instância. Uma vez retirados estes dados, é então definido o conjunto de vértices (nós) do problema, e, a partir das coordenadas dos nós, são calculadas as distâncias euclidianas entre nós, e criada a matriz de feromonas, atribuindo o valor τ_0 a todos os arcos. Esta função devolve o conjunto de vértices (nós), matriz de distâncias entre nós, capacidade máxima do veículo, matriz de procura, matriz de feromonas;

- ***solutionOfOneAnt*:** Esta função obtém a solução para uma dada formiga k. Começa pela escolha aleatória do primeiro nó para onde a formiga partirá, subtraindo, em seguida, ao valor da capacidade máxima do veículo o valor da procura no nó selecionado. Posteriormente, calcula a matriz p_{ij}, e em função das probabilidades obtidas será escolhido o próximo nó na rota da formiga k. De seguida verifica-se a condição de capacidade, e se esta condição tiver sido excedida dá-se por terminada a rota da formiga k. Esta função recebe como argumentos de entrada o conjunto de vértices, a matriz de distâncias, a capacidade máxima do veículo, a matriz de procura e a matriz de feromonas, e retorna a solução obtida para uma dada formiga k;

- ***rateSolution*:** Esta função calcula a distância total percorrida para uma dada solução para a formiga k. Os seus argumentos de entrada são a solução de uma dada formiga k, e a matriz de distâncias;

- ***updateFeromone*:** Esta função recebe como argumentos de entrada a matriz de feromonas, as soluções obtidas, e a variável *bestSolution*, devolvendo como argumento de saída a melhor solução da última iteração realizada. Esta função começa por calcular o valor de L^{avg}, e de seguinda calcular o valor de feromonas que permanecem após o processo de evaporação,

segundo a equação 8. Seguidamente, calcula os valores de $\Delta\tau_{ij}^{\lambda}$ e $\Delta\tau_{ij}*$ para calcular então a concentração de feromonas depositada, segundo a equação 9;

- ***main*:** Esta é a função principal que corre o algoritmo.

Note-se que na descrição das funções foi usada a palavra "matriz"com alguma liberdade, a sua utilização não indica necessariamente que a variável em questão é do tipo *array*. Visto que o algoritmo foi implementado em *python* a maioria das variáveis são do tipo *list*.

3.3 Resultados

Para cada uma das instâncias foram feitos ensaios com 1000 e 10000 iterações para verificar de que forma o aumento de iterações melhorava a solução para um dado conjunto fixo de parâmetros (Ensaios A). Para estes ensaios foram utilizados os seguintes parâmetros: $\alpha = 2$, $\beta = 5$, $\sigma = 3$ formigas elitistas, $\rho = 0.80$, $\theta = 80$, $ants = n$, onde n é o número de nós da respetiva instância, (escolha dos parâmetros baseada nos ensaios realizados por Tan,W.F., L.S. Lee, Z.A. Majid e H.V. Seow [3]), e $\tau_0 = 1$. Posteriormente, uma vez que são sabidas à priori o número de rotas das melhores soluções encontradas, experimentou-se, para 10000 iterações, diminuir o número de formigas artificiais para 10 nas instâncias 1 e 2, e para 20 na instância 3, e aumentar a influência da heurística na escolha dos nós em cada rota, aumentando β para 7 e diminuindo τ_0 para 0.5 (Ensaios B), e comparou-se de que forma estas mudanças influenciaram os resultados. Foram realizados 8 ensaios para cada conjunto de parâmetros.

3.3.1 1ª Instância

De seguida são apresentadas as tabelas que demonstram os resultados dos ensaios A e B. As tabelas apresentam o valor médio obtido, o melhor valor obtido, e, no caso das rotas, a percentagem de vezes que surgiu um certo número de rotas (visto que apresentar "um número médio de rotas"não faria sentido).

Ensaios A

Para os ensaios com 1000 iterações o tempo médio de computação foi de $36.22s$.

Tabela 1: Resultados para 1000 iterações (ensaio A)

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	4	377.43	0.65
Resultado Médio	4 (100%)	380.44	1.45

A tabela com todos os ensaios encontra-se em anexo. Para os ensaios A com 1000 iterações o valor 377.43 surgiu apenas uma vez, e o valor 378,66 foi o valor mais frequente, surgindo um total de 2 vezes.

Para os ensaios A com 10000 iterações o tempo médio de computação foi de $363.81s$ ($\approx$6min).

Tabela 2: Resultados para 10000 iterações (ensaio A)

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	4	377.43	0.65
Resultado Médio	4 (100%)	378.23	0.86

A tabela com todos os ensaios encontra-se em anexo. Para os ensaios A com 10000 iterações o valor 377.43 surgiu duas vezes, e o valor 378,66 foi o valor mais frequente, surgindo um total de 5 vezes.

Pode-se concluir que tanto os ensaios com 1000 como com 10000 iterações foram bem sucedidos. obtendo-se erros na ordem do 1%, e verifica-se que com o aumento do número de iterações os resultados aproximam-se do resultado ótimo, como seria de esperar.

Ensaios B

Para os ensaios B com 10000 iterações o tempo médio de computação foi de $400.61s$ ($\approx$6.5min).

Tabela 3: Resultados para 10000 iterações (ensaio B)

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	4	377.43	0.65
Resultado Médio	4 (100%)	378.51	0.94

A tabela com todos os ensaios encontra-se em anexo. Para os ensaios B com 10000 iterações o valor 377.43 surgiu apenas uma vez, e o valor 378,66 foi o valor mais frequente, surgindo um total de 7 vezes. Tendo em conta a frequência com que surgiu a solução 378,66 em todos os ensaios pode-se concluir que esta será um mínimo local, e ao diminuir o número de formigas tornou-se o algoritmo mais propicio a convergir prematuramente, o que, de facto, aconteceu.

3.3.2 2ª Instância

De seguida são apresentadas as tabelas que demonstram os resultados dos ensaios A e B. As tabelas apresentam o valor médio obtido, o melhor valor obtido, e, no caso das rotas, a percentagem de vezes que surgiu um certo número de rotas.

Ensaios A

Para os ensaios com 1000 iterações o tempo médio de computação foi de $266.42s$ ($\approx$ 4min).

Tabela 4: Resultados para 1000 iterações (ensaio A)

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	5	609.94	17.07
Resultado Médio	5 (25%) 6 (75%)	631.26	21.16

Para os ensaios A com 1000 iterações nenhum valor surgiu mais do que uma vez.

Para os ensaios com 10000 iterações o tempo médio de computação foi de $3393.51s$ ($\approx$ 56min).

Tabela 5: Resultados para 10000 iterações (ensaio A)

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	5	591.16	13.47
Resultado Médio	5 (50%) 6 (50%)	631.26	19.01

A tabela com todos os ensaios encontra-se em anexo. Para os ensaios A com 10000 iterações nenhum valor surgiu mais do que uma vez. Verifica-se que, mais uma vez, o aumento do número de iterações melhorou a solução obtida , como seria de esperar.

Ensaios B

Para os ensaios B com 10000 iterações o tempo médio de computação foi de 1432.98s ($\approx$24min).

Tabela 6: Resultados para 10000 iterações (ensaio B)

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	6	592.27	13.68
Resultado Médio	5 (12.5%) 6 (87.5%)	603.49	15.83

A tabela com todos os ensaios encontra-se em anexo. Para esta instância, ao contrário do que se observou na instância anterior, não se verificou uma convergência para nenhuma solução particular, tal dever-se-à ao facto das soluções obtidas ainda se encontrarem consideravelmente afastadas da solução ótima, levando a crer que não houve iterações suficientes para que o algoritmo tivesse sequer oportunidade de convergir para uma solução subótima. Comparando com os ensaios A, verifica-se que, embora o melhor valor encontrado em B não tenha sido melhor que o encontrado em A, no geral as soluções obtidas apresentaram distâncias menores, mesmo com um número de rotas maior. A diminuição do número de formigas também diminui o tempo de computação para menos de metade do do ensaio A com 10000 iterações.

3.3.3 3ª Instância

De seguida são apresentadas as tabelas que demonstram os resultados dos ensaios A e B. As tabelas apresentam o valor médio obtido, o melhor valor obtido, e, no caso das rotas, a percentagem de vezes que surgiu um certo número de rotas.

Ensaios A

Para os ensaios com 1000 iterações o tempo médio de computação foi de 2140.00s ($\approx$ 35min).

Tabela 7: Resultados para 1000 iterações (ensaio A)

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	14	1301.62	21.53
Resultado Médio	14 (62.5%) 15 (37.5%)	1322.47	23.48

Para os ensaios A com 1000 iterações nenhum valor surgiu mais do que uma vez.

Para os ensaios com 10000 iterações o tempo médio de computação foi de 55534.7s ($\approx$ 15.5h).

Tabela 8: Resultados para 10000 iterações (ensaio A)

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	14	1299.19	21.31
Resultado Médio	14 (87.5%) 15 (12.5%)	1307.41	22.07

A tabela com todos os ensaios encontra-se em anexo. é possível constatar que, mais uma vez, o aumento do número de iterações melhorou as solução obtidas , como se esperava.

Ensaios B

Para os ensaios B com 10000 iterações o tempo médio de computação foi de 8370.37s ($\approx$2h20min).

Tabela 9: Resultados para 10000 iterações (ensaio B)

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	14	1258.89	17.54
Resultado Médio	14 (87.5%) 15 (12.5%)	1275.23	18.79

A tabela com todos os ensaios encontra-se em anexo. Para esta instância as conclusões são muito semelhantes à instância anterior. Não se verificou uma convergência para nenhuma solução particular, tal dever-se-à ao facto das soluções obtidas ainda se encontrarem consideravelmente longe da solução ótima, levando a crer que não houve iterações suficientes para que o algoritmo tivesse oportunidade de convergir para uma solução subótima. Comparando com os ensaios A verifica-se que as soluções obtidas apresentaram distâncias menores e a diminuição do número de formigas levou a uma diminuição considerável do tempo de computação em comparação com o ensaio A com 10000 iterações.

4 Algoritmo Genético

O Algoritmo genético (*Genetic Algorithm* (GA)) é uma meta-heurística que, tal como a meta-heurística descrita anteriormente, foi inspirada em fenómenos da natureza, nomeadamente a Teoria da Evolução das Espécies formulada por Charles Darwin. Como tal, este algoritmo baseia-se em conceitos de mutação, cruzamento (*crossover*), e seleção para encontrar uma solução para problemas de otimização e pesquisa. Algoritmos genéticos permitem explorar a zona de potenciais soluções, evoluindo gradualmente de acordo com certos parâmetros pre-definidos, até se encontrar a melhor solução, sendo que não é garantido que esta seja a solução ótima.

4.1 Algoritmo Genético Aplicado ao CVRP

Primeiramente, é importante definir alguns conceitos básicos essenciais para o GA:

- **População:** A população é um conjunto de soluções possíveis para um dado problema;
- **Cromossoma:** O cromossoma ou genoma é uma solução para o dado problema;
- **Gene:** Um gene é um elemento numa dada posição do cromossoma;
- **Alelo:** O alelo é o valor que um certo gene toma para um dado cromossoma;
- **Genótipo:** O genótipo é a população no espaço computacional. No espaço computacional, as soluções são representadas de forma a que sejam facilmente interpretadas e manipuladas pelo sistema de computação;
- **Fenótipo:** O fenótipo é a população no espaço de soluções no mundo real, em que as soluções são representadas de acordo com as situações reais.

- **Descodificação e Codificação:** Para problemas simples, os espaços do fenótipo e do genótipo são os mesmos, contudo para a maior parte dos problemas este não é o caso. O processo de descodificação consiste em transformar a solução do genótipo para o fenótipo, e a codificação é o processo contrário. Para exemplificar, no problema CVRP resolvido os clientes são referidos de acordo com um número que lhes foi atribuído, mas na vida real estes cliente seriam os nomes de cidades.

Assim, tendo em conta os conceitos apresentados e a descrição do GA feita previamente, a estrutura básica de um GA é a seguinte: começa-se por inicializar a população (a inicialização poderá ser aleatória ou de acordo com alguma heurística); de seguida utiliza-se a função de aptidão (*fitness function*, ou função de custo) para calcular o valor de *fitness* de cada solução na população, e, posteriormente, selecionam-se as duas soluções que serão os parentes; aplica-se então o cruzamento (*crossover*) a estes dois genomas para obter os filhos, aos quais se aplica a mutação e estes filhos irão substituir indivíduos existentes na população. Este algoritmo pode ser escrito em pseudo-código da seguinte forma:

Algorithm 2 Genetic Algorithm

```
1: Inicialização da População
2: Calcular o valor de fitness da população
3: while Condição de paragem não é atingida do
4:     Selecionar parentes
5:     Aplicar crossover
6:     Aplicar mutação
7:     Cálculo do valor de fitness
8:     Selecionar sobreviventes
9:     Encontrar o melhor
10: Devolver o melhor
```

Nos parágrafos que se seguem são apresentadas a decisões que foram tomadas no que toca à representação da solução, à inicialização e atualização da população, à escolha dos parentes, à operação de *crossover*, à operação de mutação e condição de paragem.

4.1.1 Representação das soluções

Para uma implementação do GA bem sucedida é necessário, primeiro que tudo, decidir a representação que se irá utilizar para representar as soluções. A representação utilizada é muito característica ao problema que se pretende resolver e não há nenhuma abordagem única para a seleção da representação. Alguns exemplos de representações possíveis são a representação binária em que cada solução é um sequência de *bits*, a representação de valor real que poderá ser usada em problemas contínuos, e a representação inteira que seria indicada para um problema discreto em que as soluções não se limitam apenas a um espaço binário. Contudo, para a resolução do CVRP a representação escolhida foi a representação de cada rota de uma solução por permutações, visto que estas consistem de uma ordem de elementos.

4.1.2 Inicialização e Atualização da População

Tendo em conta os conceitos referidos previamente, um população pode ser definida como um conjunto de cromossomas (ou genomas). Há certas características que a população deverá ter para que o GA seja bem sucedido, como manter diversidade na população para prevenir a convergência prematura para uma solução subótima, e o tamanho da população deverá ser escolhido

adequadamente visto que populações muito grandes resultam em grandes tempos de computação, e populações muito pequenas poderão não ser suficientemente diversas. Para a resolução do CVRP foi utilizada uma população constituída por 30 cromossomas. Este valor foi escolhido tendo por base os testes feitos por Puljic e Manger [10].

Como mencionada acima, a população pode ser inicializada a partir de soluções geradas aleatóriamente, ou utilizando uma heurística. Na resolução deste problema a população foi gerada aleatóriamente, sendo que a única restrição foi manter o constrangimento de capacidade. Este constrangimento foi utilizado para decidir quando uma rota da solução terminava e começava outra, mas a sequência de vértices foi gerada aleatóriamente.

Relativamente a modelos de população, existem dois principais: estacionário e geracional. No modelo estacionário são gerados um ou dois filhos em cada iteração que substituem um ou dois indivíduos da população, no modelo geracional são criados n filhos, em que n é o tamanho da população, e todos os indivíduos da população são substituídos. No algoritmo criado para este projeto utilizou-se um modelo geracional com elitismo, isto é, as duas melhores soluções são guardadas para a geração seguinte e apenas são gerados $n-2$ filhos, que substituem os outros $n-2$ indivíduos em cada iteração.

4.1.3 Escolha dos Parentes

Um bom método de seleção de parentes é crucial para manter a diversidade na população e controlar a taxa de convergência do GA, impedindo convergência precoce. A escolha dos cromossomas parentes neste trabalho foi feita por *fitness propotionate selection*, ou seja, qualquer indivíduo pode-se tornar num parente mas a probabilidade de tal acontecer é proporcional ao seu valor de *fitness*. Assim as soluções mais aptas (*fitter solutions*) têm uma maior probabilidade de transmitirem as suas características à próxima geração.

4.1.4 Cruzamento

O cruzamento, ou *crossover* permite gerar novas soluções recombinando os cromossomas parentes, semelhantemente com o que se observa na natureza. Existem diversos operadores de *crossover* que se podem aplicar, podendo um algoritmo até utilizar mais do que um. Neste projecto, para resolver o CVRP, optou-se pela utilização do *alternating edges crossover* (AEX), tendo por base os testes realizados por Puljic e Manger [10], onde este operador demonstrou um desempenho muito bom para a resuloção do CVRP. O AEX gera um cromossoma filho escolhendo arcos de cada um dos parentes alternadamente.

4.1.5 Mutação

Uma mutação é uma pequena mudança aleatória no cromossoma, resultando numa solução nova. A mutação é o principal mecanismo do GA que mantém a diversidade de soluções e permite a exploração de novos espaços de soluções, ajudando a evitar mínimos (ou máximos em problemas de maximização) locais. No GA criado para este trabalho utilizaram-se três métodos de mutação com a mesma probabilidade de serem selecionados numa dada iteração. Os métodos usados foram: *Swap Mutation*, em que duas entradas, escolhidas aleatoriamente, de uma das rotas da solução, escolhida aleatoriamente, trocam de posição entre si; *Scramble Mutation*, em que as entradas de uma das rotas, escolhida aleatoriamente, são "desorganizadas"criando uma nova solução; e *Inverse Mutation* em que para uma dada rota aleatória são selecionadas duas das suas entradas, e todas as entradas entre essas duas selecionadas são recolocadas invertendo a sua ordem. A utilização de

três métodos de mutação diferentes em simultâneo foi também inspirada pelo trabalho de Puljic e Manger [10], embora os métodos utilizados não tenham sido exatamente os mesmos.

4.1.6 Condição de Paragem

A condição de paragem deve ser definida de forma a que quando o GA pare de correr se tenha obtido uma solução muito perto da solução ótima. Neste trablho o único critério de paragem foi definido pelo número máximo de gerações (ou iterações).

4.2 Implementação Computacional

A implementação do GA foi feita na linguagem de programação *python*, encontrando-se no ficheiro *GA_CVRP.py*, e para correr este ficheiro utilizou-se o software *PyCharm Community Edition 2020.3.3*.

Uma vez aberto o ficheiro, o utilizador deverá escrever o nome do ficheiro *.txt* que contem os dados da instância que pretende correr (variável *fileName*), o número máximo de iterações (variável *max_generations*), e o tamanho da população (variável *populationSize*). De seguida, o utilizador poderá correr o código.

Para implementação do GA foram criadas as seguintes funções:

- ***get_data*:** Esta função lê o ficheiro *.txt* e devolve a matriz de vértices do problema, a matriz de distancias entre vértices, a capacidade máxima do veículo, a matriz de procura, e o melhor valor encontrado na literatura para a instância (*optimalValue*). Para ler o documento recorreu-se ao mesmo ficheiro utilizado para o ACO, *RegExService.py*.

- ***generate_genome*:** esta função recebe como argumentos de entrada a matriz de vértices, a matriz de procura e a capacidade máxima do veículo, e devolve um cromossoma gerado aleatóriamente (*genome*). Esta função começa por criar a lista *genome* e a variável *city* do tipo *array*, preenchida com zeros. Até que que não haja mais vértices por visitar, são atribuídos vértices às entradas da variável *city* aleatoriamente. Em cada ciclo é verificada condição de capacidade. Se a adição do novo vértice à rota não violar esta condição o ciclo continua na mesma rota, e o vértice é adicionado à rota e removido da lista de vértices não visitados. Se a condição de capacidade tiver sido violada é começada uma nova rota, e a rota anterior é adicionada à lista *genome*.

- ***generate_population*:** Esta função recebe como *input* a dimensão pretendida da população, a matriz de vértices, a matriz de procura e a capacidade máxima do veículo, e devolve a população. Esta função é utilizada para inicializar a população.

- ***fitness_func*:** Esta função recebe um cromossoma (genoma) e a matriz de distâncias entre nós, e devolve o valor de *fitness* o mesmo cromossoma.Este valor é calculado de acordo com a função de custo apresentada na secção 2, ou seja, somando as distâncias entre os vértices consecutivos nas rotas.

- ***selection_pair*:** Esta função seleciona os cromossomas que serão os parentes da próxima geração de soluções. Esta função serve-se da função *fitness_func* para calcular o valor de fitness para todas as soluções da população, e depois escolhe aleatoriamente 2 cromossomas. Esta escolha é feita recorrendo à função *choices* da biblioteca *random*, em que se definiu que o parâmetro *weights* é igual ao inverso do erro relativo do valor de *fitness* do genoma. A

implementação foi feita desta forma de maneira a dar prioridade aos valores menores. Os argumentos de entrada desta função são a população, a matriz de distâncias entre nós, e o melhor valor conhecido, e os argumentos de saída são os genomas parentes.

- ***alternating_edges_crossover*:** Esta função aplica o operador de *crossover* descrito anteriormente. Primeiramente, a função guarda numa lista todos os nós correspondentes ao inicio de rotas de ambos os parentes para serem também usados no inicio das rotas do cromossoma filho. A função começa então a construir o cromossoma filho, atribuindo como primeiro vértice da rota, um dos primeiros vértices das rotas do parente 1, sendo esse vértice posteriormente retirado da lista de vértices potenciais e da lista dos vértices iniciais dos parentes.

  De seguida, procura-se na solução do parente 2 a localização da primeira entrada do cromossoma filho. Se esta for localizada, o vértice seguinte na rota do parente 2 será também o vértice seguinte do filho. Se não for localizada, o próximo vértice do filho é escolhido aleatoriamente a partir da lista de potenciais vértices.

  Depois é necessário verificar a condição de capacidade. Se a capacidade não tiver sido excedida o novo vértice é retirado da lista de vértices potenciais e vai-se buscar o próximo arco ao parente 1 (caso o último arco se tenha obtido do parente 2; se o último vértice do filho tiver sido obtido aleatoriamente, o próximo arco será retirado do parente 2). Se a capacidade for excedida começa-se uma nova rota. O primeiro vértice da nova rota será um dos primeiros vértices de rotas do parente 2, ou será gerado aleatoriamente se os vértices iniciais deste parente já tiverem sido todos visitados.

  O ciclo repete-se até que a lista de vértices possíveis esteja vazia. Esta função recebe como argumento de entrada os dois genomas parentes, a matriz de vértices, a matriz de procura, a capacidade máxima do veículo, e uma variável booleana que indica em qual dos parentes se começa a ir buscar entradas (0 corresponde ao *genome_a* e 1 corresponde ao *genome_b*).

- ***mutation_func*:** Esta função aplica um operador de mutação ao genoma (argumento de entrada), devolvolvendo o genoma modificado. Como explicado previamente, esta função aplica um de três operadores diferente de mutação (*Swap*, *Scramble* ou *Inverse*) com igual probabilidade. A a função começa por definir a variável *p* com um número escolhido aleatoriamente entre 0, 1 ou 2, sendo que 0 corresponde a aplicar *Swap Mutation*, 1 corresponde a aplicar *Scramble Mutation*, e 2 corresponde a aplicar *Inverse Mutation*.

- ***run_evolution*:** Esta é a função principal, é a função que corre o algoritmo genético de acordo com o pseudo-código apresentado previamente no algoritmo 2.

Note-se que na descrição das funções foi usada a palavra "matriz"com alguma liberdade, a sua utilização não indica necessariamente que a variável em questão é do tipo *array*. Visto que o algoritmo foi implementado em *python* a maioria das variáveis são do tipo *list*.

4.3 Resultados

Para cada uma das instâncias foram feitos oito ensaios com 1000 e 10000 iterações para verificar de que forma o aumento de iterações melhorava a solução. Como mencionado anteriormente, o tamanho da população foi definido como igual a 30.

4.3.1 1ª Instância

Nas tabelas 10 e 11 são apresentados o valor médio obtido, o melhor valor obtido, e, no caso das rotas, a percentagem de vezes que surgiu um certo número de rotas para os ensaios com 1000 e 1000 iterações respetivamente.

Para os ensaios com 1000 iterações o tempo médio de computação foi de $40.14s$.

Tabela 10: Resultados obtidos pelo GA para 1000 iterações para a 1ª instância

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	4	375.28	0.075
Resultado Médio	4 (100%)	407.65	0.087

Para os ensaios com 10000 iterações o tempo médio de computação foi de $423.56s$ ($\approx$ 7min).

Tabela 11: Resultados obtidos pelo GA para 10000 iterações para a 1ª instância

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	4	375.28	0.075
Resultado Médio	4 (100%)	382.46	1.99

Ambas as tabelas com todos os ensaios encontram-se em anexo. Relembrando os resultados obtidos para esta instância no projeto anterior com o método exato (dados de verificação do método exato) é possível afirmar que tanto para 1000 como para 10000 iterações o GA alcançou a solução ótima (todos os valores de referência encontram-se arredondados às unidades). Mais uma vez, também se verificou que a o aumento do número de iterações melhorou as soluções obtidas, como seria de esperar. Também se observou que o valor 383.52 surgiu em 50% dos ensaios com 10000, o que levar a crer que este valor poderá ser um mínimo local.

4.3.2 2ª Instância

Nas tabelas 12 e 13 são apresentados o valor médio obtido, o melhor valor obtido, e, no caso das rotas, a percentagem de vezes que surgiu um certo número de rotas para os ensaios com 1000 e 1000 iterações respetivamente.

Para os ensaios com 1000 iterações o tempo médio de computação foi de $126.72s$ ($\approx$ 2min).

Tabela 12: Resultados obtidos pelo GA para 1000 iterações para a 2ª instância

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	5	966.65	85.54
Resultado Médio	5 (75%) 6 (25%)	1068.42	105.07

Para os ensaios com 10000 iterações o tempo médio de computação foi de $1619.11s$ ($\approx$ 27min).

Tabela 13: Resultados obtidos pelo GA para 10000 iterações para a 2ª instância

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	5	740.83	42.19
Resultado Médio	5 (87.5%) 6 (12.5%)	781.32	49.97

O aumento do número de iterações melhorou significativamente os resultados, contudo verifica-se que o número de iterações escolhido ainda foi muito baixo visto que os resultados não são de todo satisfatórios. Perante estes resultados experimentou-se fazer ensaios com 50000 e 100000 iterações, com o objetivo de perceber quantas iterações seriam precisas para se obterem resultados com erros na mesma ordem dos erros obtidos com o ACO. O melhor resultado obtido foi para 100000 iterações e foi igual a 687.16, apresentando um erro relativo de 31.89%, o qual ainda se encontra muito atrás dos piores resultados do ACO para a mesma instância.

4.3.3 3ª Instância

Nas tabelas 14 e 15 são apresentados o valor médio obtido, o melhor valor obtido, e, no caso das rotas, a percentagem de vezes que surgiu um certo número de rotas para os ensaios com 1000 e 1000 iterações respetivamente.

Para os ensaios com 1000 iterações o tempo médio de computação foi de $706.85s$ ($\approx$ 12min).

Tabela 14: Resultados obtidos pelo GA para 1000 iterações para a 3ª instância

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	14	2684.49	150.65
Resultado Médio	14 (75%) 15 (25%)	2801.19	161.55

Para os ensaios com 10000 iterações o tempo médio de computação foi de $7183.5s$ ($\approx$ 2h).

Tabela 15: Resultados obtidos pelo GA para 10000 iterações para a 3ª instância

	Número de rotas	Distância (km)	Erro (%)
Melhor Resultado	14	2050.35	91.44
Resultado Médio	14 (87.5%) 15 (12.5%)	2219.53	107.24

Mais um vez, o aumento do número de iterações melhorou significativamente os resultados, como esperado, contudo o número de iterações escolhido ainda foi muito baixo visto que os resultados não são de todo satisfatórios. Perante estes resultados experimentou-se fazer ensaios com 50000 e 100000 iterações, contudo o tempo de computação prolongou-se consideravelmente e não foi possível obter resultados conclusivos. Referenciando mais uma vez Puljic e Manger [10], no seu trabalho conseguiram obter resultados satisfatórios utilizando o AEX para instâncias de um problema de CVRP de dimensões semelhantes às apresentadas nas instâncias 2 e 3 deste relatório, contudo nos ensaios efetuados o algoritmo correu durante 4000000 iterações. Note-se que o algoritmo apresentado por Puljic e Manger [10], e o algoritmo apresentado neste trabalho não são completamente semelhantes (divergem, por exemplo, nos operadores de mutação utilizados, como já referido), contudo o cerne da questão da disparidade entre a qualidade de resultados é o número

de iterações. De facto, já foi observado que inicialmente o GA obtém cada vez melhores soluções em poucas iterações, mas isto tende a saturar nos estágios posteriores, onde as melhorias são muito pequenas.

Todas as tabelas experimentais completas encontram-se em anexo.

5 Comparação de Resultados

Nesta secção são apresentados e comparado os melhores resultados obtidos com o ACO e os resultados obtidos com o GA. Estes encontram-se na tabela 16.

Tabela 16: Comparação dos melhores resultados para o ACO e para o GA

	1ª Instância		2ª instância		3ª instância	
	ACO	GA	ACO	GA	ACO	GA
Melhor Resultado (km)	377.43	375.28	591.16	740.83	1258.89	2050.35
Erro (%)	0.65	0.075	13.47	42.19	17.54	91.44
Tempo de Computação	36.22s	40.14s	∼56min	∼27min	∼2h20min	∼2h

Analisando os resultados obtidos com ambas as metaheurísticas, facilmente se conclui que, de facto, o ACO provou ser a melhor abordagem para obter soluções satisfatórias para o CVRP em tempos de computação razoáveis. Embora o GA tenha chegado à solução ótima na primeira instância, à medida que o número de nós no problema foi aumentando, mais tempo era necessário para o GA obter soluções com erros relativos que fossem sequer da ordem de grandeza dos erros das soluções obtidas utilizando o ACO.

6 Conclusão

Neste relatório foram analisadas duas metaheuristicas, Otimização por Colónia de Formigas (*Ant Colony Optimization* (ACO)) e Algortimo Genético (*Genétic Algorithm* (GA)), e comparado o seu desempenho na resolução de um problema de *Capacitated Vehicle Routing Problem* (CVRP). Ambos os algoritmos podem ser utilizados para resolver este problema, sendo que o GA é um método que pode ser utilizado para resolver problemas discretos ou contínuos, e é mais frequentemente usado em problemas de maximização, enquanto que o ACO é um método discreto e mais utilizado em problemas de minimização. Tendo em conta estas características é fácil compreender a razão pela qual o ACO tem vindo a substituir o GA na obtenção de soluções para certos problemas, nomeadamente, neste relatório o ACO provou obter melhores resultados que o GA para as instâncias do problema CVRP testadas.

Referências

[1] F. Hillier and G. Lieberman, *Introduction to Operations Research*, 8ª Edição McGrawHill, 2005.

[2] Dorigo, M. and T. Stutzle, *Ant Colony Optimization*, 1st Edn., MIT Press, Cambridge, 2004.

[3] Tan,W.F., L.S. Lee, Z.A. Majid and H.V. Seow, *Ant Colony Optimization for Capacitated Vehicle Routing Problem*, Institute for Mathematical Research, Department of Mathematics, Faculty of Science, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia Nottingham University Business School, Faculty of Arts and Social Sciences, University of Nottingham Malaysia Campus, Jalan Broga, 43500 Semenyih, Selangor, Malaysia, 2012.

[4] Ruey-Maw Chen, Fu-Ren Hsieh , Di-Shiun Wu, *Heuristics Based Ant Colony Optimization for Vehicle Routing Problem* , Department of Computer Science and Information Engineering, NCUT, Taichung, Taiwan.

[5] Petr Stodola, Jan Mazal, Milan Podhorec and Ondřej Litvaj, *Using the Ant Colony Optimization Algorithm for the Capacitated Vehicle Routing Problem*, University of Defence, Kounicova 65, Brmo, Czech Republic.

[6] Udom Janjarassuk, Ruedee Masuchun, *An Ant Colony Optimization Method for the Capacitated Vehicle Routing Problem with Stochastic Demands*, Faculty of Engineering King Mongkut's Institute of Technology Ladkrabang Bangkok, Thailand.

[7] J.-P. Rodrigue, C. Comtois, and B. Slack, *The geography of transport systems*, Routledge, 2013.

[8] G. B. Dantzig, and J. H. Ramser, *The Truck Dispatching Problem*, Management Science, vol. 6, no. 1, pp. 80-91, 1959.

[9] Bullnheimer, B., R.F. Hartl and C. Strauss, *An improved Ant System algorithm for the Vehicle Routing Problem.*, Institute of Management Science, University of Vienna, Bruenner Str. 72, A - 1210 Vienna, Austria, 1999

[10] Krunoslav Puljic, and Robert Manger, *Comparison of eight evolutionary crossover operators for the vehicle routing problem*, Department of Mathematics, University of Zagreb, Bijeni˘cka cesta 30, HR-10 000 Zagreb, Croatia, 2011

[11] Genetic Algorithm - Wikipedia,
`https://en.wikipedia.org/wiki/Genetic_algorithm`

[12] Genetic Algorithms Tutorial - Tutorials Point,
`https://www.tutorialspoint.com/genetic_algorithms/index.htm`

[13] Mostapha Kalami Heris, Practical Genetic Algorithms in Python and MATLAB – Video Tutorial, Yarpiz, 2020.
`https://yarpiz.com/632/ypga191215-practical-genetic-algorithms-in-python-and-matlab)`

[14] Mostapha Kalami Heris, Practical Genetic Algorithms in Python and MATLAB – Video Tutorial, Yarpiz, 2020.
`https://yarpiz.com/632/ypga191215-practical-genetic-algorithms-in-python-and-matlab)`

[15] Tutorial 13: Multi-Vehicle Routing with Time Windows - Day 4 - Thursday, July 26 - Youtube,
https://www.youtube.com/watch?v=BZA$_U$aX8rs8

[16] Vehicle Routing Data Sets, Instances of Eilon and Christofides,
https://www.dca.fee.unicamp.br/projects/infobiosys/vrp/

[17] Capacitated Vehicle Routing Problem solved with Ant Colony Optimization,
https://github.com/pkonowrocki/CVRP_ACO

[18] The VRP Web,
http://www.bernabe.dorronsoro.es/vrp/

[19] Networking and Emerging Optimization,
https://neo.lcc.uma.es/vrp/known-best-results/

[20] VRP-REP,
http://www.vrp-rep.org/references/item/lysgaard-et-al-2004.html

[21] Genetic Algorithms Explained By Example - Youtube,
https://www.youtube.com/watch?v=uQj5UNhCPuot=0s

[22] Genetic Algorithm from Scratch in Python (tutorial with code) - Youtube,
https://www.youtube.com/watch?v=nhT56blfRpE

A Tabelas com Valores Experimentais

A.1 ACO

A.1.1 1ª Instância

Ensaios A

Tabela 17: Resultados para 1000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	4	378.93	1.05
2	4	383.84	2.36
3	4	378.66	0.98
4	4	378.66	0.98
5	4	383.84	2.36
6	4	381.58	1.76
7	4	377.43	0.65
8	4	380.61	1.50
Valores Médios	-	380.44	1.45

Tabela 18: Resultados para 10000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	4	378.66	0.98
2	4	378.66	0.98
3	4	378.66	0.98
4	4	377.43	0.65
5	4	377.43	0.65
6	4	377.69	0.72
7	4	378.66	0.98
8	4	378.66	0.98
Valores Médios	-	378.23	0.86

Ensaios B

Tabela 19: Resultados para 10000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	4	378.66	0.98
2	4	377.43	0.65
3	4	378.66	0.98
4	4	378.66	0.98
5	4	378.66	0.98
6	4	378.66	0.98
7	4	378.66	0.98
8	4	378.66	0.98
Valor Médio	-	378.51	0.94

A.1.2 2ª Instância

Ensaios A

Tabela 20: Resultados para 1000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	6	636.64	22.20
2	6	635.64	22.00
3	5	609.94	17.07
4	6	633.06	21.51
5	6	636.82	22.23
6	6	630.50	21.02
7	5	628.39	20.61
8	6	639.11	22.67
Valores Médios	-	631.26	21.16

Tabela 21: Resultados para 10000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	6	617.23	18.47
2	6	607.53	16.61
3	5	619.52	18.91
4	5	622.66	19.51
5	6	612.48	17.56
6	5	610.23	17.13
7	5	591.16	13.47
8	6	599.84	15.13
Valores Médios	-	610.08	17.10

Ensaios B

Tabela 22: Resultados para 10000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	6	599.48	15.06
2	6	603.07	15.75
3	6	609.03	16.90
4	5	610.26	17.13
5	6	592.27	13.68
6	6	604.89	16.10
7	6	605.34	16.19
8	6	603.54	15.84
Valor Médio	-	603.49	15.83

A.1.3 3ª Instância

Ensaios A

Tabela 23: Resultados para 1000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	14	1327.33	23.93
2	14	1316.04	22.88
3	14	1301.62	21.53
4	14	1333.67	24.53
5	14	1322.26	23.46
6	14	1330.68	24.25
7	15	1313.56	22.65
8	15	1334.60	24.61
Valores Médios	-	1322.47	23.48

Tabela 24: Resultados para 10000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	14	1301.11	21.49
2	14	1307.73	22.10
3	14	1314.31	22.72
4	14	1310.43	22.36
5	14	1311.00	22.41
6	14	1307.40	22.07
7	14	1299.19	21.31
8	15	1308.11	22.14
Valor Médio	-	1307.41	22.07

Ensaios B

Tabela 25: Resultados para 10000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	14	1258.89	17.54
2	14	1283.86	19.88
3	14	1272.28	18.79
4	14	1263.67	17.99
5	14	1281.25	19.63
6	15	1264.89	18.10
7	14	1278.04	19.33
8	14	1274.92	19.04
Valor Médio	-	1272.23	18.79

A.2 GA

A.2.1 1ª Instância

Tabela 26: Resultados para 1000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	4	412.094307	9.89
2	4	453.758439	21.00
3	4	375.279787	0.075
4	4	405.8377	8.22
5	4	383.51683	2.27
6	4	393.22062	4.86
7	4	446.881408	19.17
8	4	390.636244	4.17
Valor Médio	-	407.653167	8.71

Tabela 27: Resultados para 10000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	4	376.51	0.40
2	4	375.28	0.075
3	4	386.92	3.18
4	4	383.52	2.27
5	4	383.52	2.27
6	4	386.92	3.18
7	4	383.52	2.27
8	4	383.52	2.27
Valor Médio	-	382.46	1.99

A.2.2 2ª Instância

Tabela 28: Resultados para 1000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	5	1046.69	100.90
2	5	1164.99	123.61
3	5	1019.07	95.60
4	6	1141.40	119.08
5	5	1107.75	112.62
6	5	1050.74	101.68
7	5	966.65	85.54
8	6	1050.09	101.55
Valor Médio	-	1068.42	105.07

Tabela 29: Resultados para 10000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	5	796.62	52.90
2	6	740.83	42.19
3	5	795.55	52.70
4	5	746.04	43.19
5	5	752.64	44.46
6	5	757.25	45.34
7	5	823.97	58.15
8	5	837.68	60.78
Valor Médio	-	781.32	49.97

A.2.3 3ª Instância

Tabela 30: Resultados para 1000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	15	2818.70	163.18
2	14	2904.34	171.18
3	14	2806.46	162.04
4	14	2811.58	162.52
5	14	2761.40	157.83
6	14	2684.49	150.65
7	14	2753.59	157.10
8	15	2868.99	167.88
Valor Médio	-	2801.19	161.55

Tabela 31: Resultados para 10000 iterações

Número do Ensaio	Número de Rotas	Distância Percorrida [km]	Erro [%]
1	15	2169.83	102.60
2	14	2128.14	98.71
3	14	2255.34	110.58
4	14	2291.22	113.93
5	14	2299.18	114.68
6	14	2123.64	98.29
7	14	2050.35	91.44
8	14	2438.52	127.69
Valor Médio	-	2219.53	107.24