# Relatório do Projeto 3

Projeto desenvolvido durante o 2º semestre do ano letivo de 2022/2023.

Computação Inteligente para a Internet das Coisas

1st Gustavo Caria
*MSc em Matemática Aplicada e Computação*
*Nº 92633*
Lisboa, Portugal
gustavocaria@sapo.pt

2nd Leonardo Brito
*MSc em Eng. e Ciência de Dados*
*Nº 105257*
Lisboa, Portugal
leonardo.amado.brito@tecnico.ulisboa.pt

***Abstract*—Neste estudo, apresentamos a implementação de um sistema avançado de recolha de materiais recicláveis no Município de Oeiras, em Portugal. O sistema é composto por 100 Ecopontos inteligentes, equipados com sensores para detetar o seu nível de enchimento. Utilizando Algoritmos Genéticos (AG), desenvolvemos um sistema de rotas inteligente que cria novas rotas diárias para o camião de recolha de resíduos. O objetivo é otimizar a recolha, considerando a capacidade dos EcoPontos e minimizando o tempo de planeamento. Apresentamos os resultados do sistema, incluindo a rota mais curta que abrange todos os EcoPontos. A implementação deste sistema inteligente destaca o potencial de transformar tarefas de rotina em processos automatizados e eficientes, proporcionando benefícios económicos e aumento da produtividade.**

***Index Terms*—Ecopontos inteligentes, algoritmos genéticos, otimização, produtividade, benefícios económicos.**

## I. Introdução

O Município de Oeiras, em Portugal, está a implementar um sistema avançado de recolha de materiais recicláveis, com o objetivo de melhorar a eficiência e a sustentabilidade do processo de reciclagem. Este relatório apresenta os principais elementos desse sistema inteligente, incluindo a distribuição de 100 EcoPontos por todo o município e a utilização de Algoritmos Genéticos para otimizar as rotas de recolha.

Os EcoPontos são pontos de recolha equipados com sensores que detetam o nível de enchimento dos contentores de reciclagem. Através de comunicação sem fios, os EcoPontos enviam informações para um Centro de Controlo, indicando quando necessitam de ser esvaziados. Essa comunicação em tempo real permite uma gestão mais eficiente dos recursos e evita recolhas desnecessárias.

Um dos principais desafios enfrentados pelo município é a otimização das rotas de recolha. O objetivo é encontrar a rota mais curta que permita recolher os materiais recicláveis de todos os EcoPontos, tendo em conta as restrições de capacidade e o tempo disponível para o planeamento. Para resolver este problema complexo, são aplicados Algoritmos Genéticos, que permitem encontrar soluções próximas da ótima num curto espaço de tempo.

Neste relatório, descrevemos o processo de desenvolvimento e implementação deste sistema inteligente. Apresentamos exemplos de resultados obtidos com diferentes configurações e discutimos a eficácia do sistema em termos de tempo de execução e comprimento da rota. Além disso, destacamos os benefícios económicos e ambientais que este sistema traz para o Município de Oeiras.

Este estudo visa demonstrar o potencial das soluções inteligentes na otimização da gestão de resíduos e na criação de cidades mais sustentáveis. Ao utilizar tecnologias avançadas, como Algoritmos Genéticos, é possível melhorar a eficiência da recolha de materiais recicláveis, reduzir os custos operacionais e promover a preservação do meio ambiente. O relatório conclui com recomendações para futuras melhorias e expansão do sistema, visando alcançar uma gestão de resíduos ainda mais eficiente e sustentável no Município de Oeiras.

## II. Escolhas no *SETUP*

Utilizámos a *framework DEAP* - por ter sido usada durante as aulas práticas - as escolhas, criação de funções e de classes foram relativas a ela.

### A. Função Fitness

Quantifica a qualidade ou desempenho das soluções individuais dentro da população. A função de avaliação atribui uma pontuação a cada solução candidata com base em quão bem ela satisfaz os objetivos ou restrições do problema. No nosso caso, queremos minimizar a distância total percorrida, portanto, a nossa função será a distância total da rota proposta pelo algoritmo.

### B. Classes

Precisámos de criar duas classes: FitnessMin e Individual.

FitnessMin: Esta classe é utilizada para representar a aptidão dos indivíduos no algoritmo evolutivo. Neste caso, a FitnessMin indica que o objetivo é minimizar o valor da Fitness. O *weights* é definido como (-1.0,), indicando que o valor deve ser minimizado.

Individual: Esta classe representa um indivíduo na população e é essencialmente uma lista de valores que codificam uma solução para o problema em questão. Neste caso, um Indivíduo é uma lista de inteiros que representam os indíces dos ecopontos.

### C. Registrations

Definimos um conjunto de operadores genéticos e funções utilizando a *toolbox* da biblioteca *DEAP*. Esses registos são essenciais para definir o comportamento do algoritmo evolutivo e permitir que ele execute as operações genéticas necessárias nos indivíduos dentro da população.

- "indices" - é utilizado um método de permutação para gerar uma amostra aleatória de índices únicos a partir do intervalo de números que representa o número de ecopontos.
- "individual" - é gerado um indivíduo com base na classe "Individual" que foi previamente definida. Utiliza-se a função "indices" para gerar uma ordem aleatória de índices para o indivíduo.
- Registo de uma função chamada "*population*" que gera uma população de indivíduos com base na classe "list". Utiliza-se a função "*individual*" para criar cada indivíduo dentro da população.
- Registo de uma função chamada "mate" que realiza o *crossover* ordenado entre dois indivíduos pais.
- Registo de uma função chamada "*mutate*" que realiza a mutação por baralhamento num indivíduo. A mutação por baralhamento é um operador genético que baralha aleatoriamente a ordem dos índices dentro de um indivíduo com uma probabilidade definida pelo programador.
- Registo de uma função chamada "*select*" que seleciona por torneio na população.
- Registo de uma função chamada "*evaluate*" que utiliza a função *evalTSP* para avaliar a *fitness* de um indivíduo, onde é calculada a distância total da rota de um indivíduo.

O torneio, a mutação por baralhamento e o *crossover* ordenado foram escolhidos como as melhores opções para o problema por várias razões.

Torneio:

- Vantagens: O torneio permite um equilíbrio entre a exploração de novas áreas do espaço de solução e a exploração de soluções já encontradas.
- Alternativas: Outras técnicas de seleção populares incluem seleção por roleta e seleção por classificação. No entanto, a seleção por roleta pode levar à predominância de soluções de alta aptidão e a seleção por classificação pode não acrescentar diversidade suficiente.

Mutação por baralhamento:

- Vantagens: A mutação por baralhamento ajuda a manter a diversidade dentro da população e pode evitar que o algoritmo fique preso em ótimos locais. A taxa de mutação escolhida é razoável, não sendo muito alta para causar mudanças disruptivas, nem muito baixa para se tornar ineficaz.
- Alternativas: Outras técnicas de mutação incluem a mutação por troca de genes e a mutação por inversão. A mutação por troca de genes pode não garantir diversidade suficiente em problemas de permutação, e a mutação por inversão pode levar a mudanças muito drásticas.

*Crossover* ordenado:

- Vantagens: O *crossover* ordenado é uma escolha sólida para problemas de permutação. Tende a criar descendentes que herdam uma boa combinação da ordem das cidades (genes) dos pais, o que muitas vezes resulta em uma boa solução se os pais também forem bons.
- Alternativas: Outras técnicas de recombinação incluem o *partially mapped crossover* e o *position crossover*. No entanto, *partially mapped crossover* pode levar a problemas de violação de restrições nalgumas situações, e o *position crossover* pode não garantir uma boa exploração do espaço de solução em problemas de permutação.

### D. Ordered Crossover

O *crossover* ordenado é um operador genético utilizado em algoritmos evolucionários para criar novos descendentes a partir de indivíduos pais. Este operador é frequentemente utilizado em problemas que envolvem permutações, como o problema do *travelling salesman* [1].

O funcionamento do *crossover* ordenado é o seguinte:

1. Selecionam-se dois indivíduos pais da população. Cada pai representa uma possível solução codificada como uma permutação de elementos.
2. Uma posição de *crossover* é selecionada aleatoriamente. Este ponto define uma secção dos cromossomas dos indivíduos pais.
3. O material genético dentro da secção de *crossover* é copiado de um dos pais para a descendência, mantendo a ordem de aparecimento no pai.
4. As posições restantes no cromossoma da descendência são preenchidas com o material genético do segundo pai, preservando a ordem e excluindo quaisquer duplicados.
5. A descendência resultante herda características de ambos os pais, combinando a ordem de aparecimento de um pai com o material genético do outro pai.
6. A nova descendência pode então ser avaliada e pode passar por outras operações genéticas, como a mutação, ou ser selecionada para formar a próxima geração.

O *crossover* ordenado aproveita a ordem de aparecimento dos elementos nos indivíduos pais para preservar informações estruturais importantes. Isto ajuda a manter bons blocos sequenciais ou subestruturas de ambos os pais na descendência, permitindo exploração do espaço de procura.

Ao aplicar o *crossover* ordenado iterativamente ao longo de várias gerações, o algoritmo evolutivo pode explorar o espaço de busca e potencialmente convergir para soluções melhores para o problema em questão.

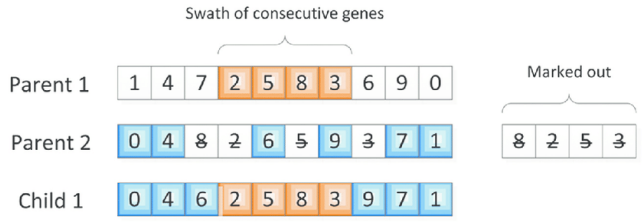


Fig. 1: Order Crossover example.

### E. Shuffle Mutation

A mutação por baralhamento é um operador genético utilizado nos algoritmos evolutivos, como algoritmos genéticos, para introduzir variação nas soluções da população. Essa mutação é frequentemente aplicada em problemas que envolvem permutações [2].

A mutação por baralhamento funciona da seguinte maneira:

- Pontos Ordenados:
  1. Seleciona-se um indivíduo da população para sofrer a mutação.
  2. Escolhe-se um ponto de mutação ou uma posição de mutação dentro do cromossoma do indivíduo.
  3. Os elementos no cromossoma a partir do ponto de mutação são baralhados aleatoriamente, criando uma nova ordem de elementos.
  4. O cromossoma do indivíduo é atualizado com a nova ordem gerada pela mutação.
  5. O indivíduo *mutado* é adicionado à população para continuar o processo evolutivo.

A mutação por baralhamento introduz diversidade e exploração no espaço de busca, permitindo que a população explore diferentes combinações de elementos. Isso ajuda a evitar a convergência prematura e a procura de soluções potencialmente melhores.

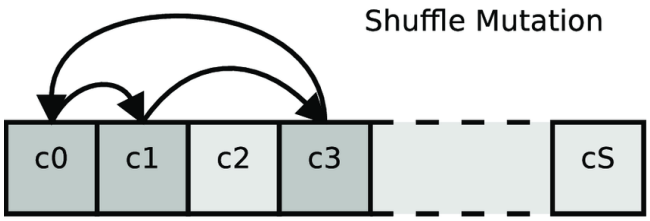


Fig. 2: Shuffle Mutation example.

### F. Seleção Tournament

A seleção por torneio é um mecanismo de seleção utilizado em algoritmos genéticos para escolher indivíduos promissores para reprodução e formação da próxima geração. Nessa abordagem, um subconjunto de indivíduos é selecionado aleatoriamente da população e competem entre si. O indivíduo com o melhor desempenho dentro desse subconjunto é escolhido como o vencedor do torneio e é selecionado para reprodução [3].

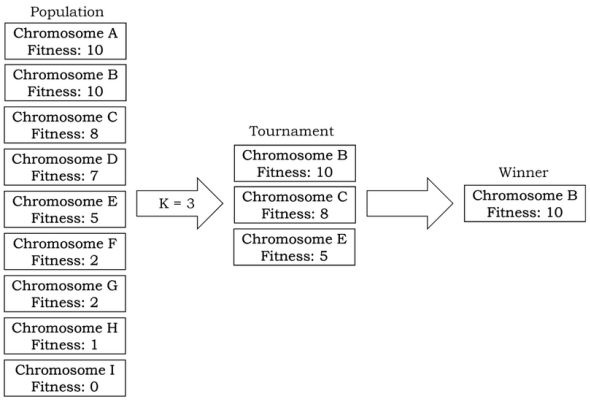


Fig. 3: Seleção por torneio onde um número maior de *fitness* é melhor que um menor.

## III. Análise de Resultados

Nós definimos, como *setup* base, uma seleção por torneio com tamanho 3, *crossover* ordenado, mutação por baralhamento de 5%, uma população de 1000 indivíduos e 10000 gerações. O número de gerações em um algoritmo genético é geralmente definido como 10 vezes maior que o tamanho da população. Isso permite que o algoritmo explore um espaço de busca maior e tenha mais oportunidades de encontrar soluções ótimas ou de alta qualidade ao longo do tempo. Com mais iterações, o algoritmo tem mais chances de aprimorar as soluções encontradas. Em seguida, nós alterámos uma variável e mantivemos as outras predefinidas a cada vez que realizamos testes. Tínhamos dois critérios de paragem: um limite de tempo predefinido, 19 minutos, e o outro era comparar a fitness (i.e. distância total) do melhor indivíduo da geração atual com o melhor de várias gerações atrás (dizemos ”várias” porque este número dependia do número de gerações total). Isso é útil quando o algoritmo precisa operar dentro de uma restrição de tempo específica, e quando nenhuma melhoria é observada em um número de gerações consecutivas, ou seja, se a *fitness* do melhor indivíduo na população não melhorar por um certo número de gerações, o algoritmo termina.

Agora vamos explicar o significado do próximo gráfico (figura 4). É importante, visto que vamos ter vários destes ao longo do relatório).

Portanto, o gráfico da figura 4 trata-se da evolução da *fitness* (i.e. distância total do caminho percorrido) de certos indivíduos a cada geração:

- a roxo, a evolução da *fitness* do **pior** indivíduo (i.e. caminho) de cada geração
- a verde, o mesmo, para o indivíduo **médio**
- a vermelho, o mesmo, para o **melhor** indivíduo de cada geração
- a azul, a evolução da *fitness* do melhor indivíduo final, desde que ”nasceu”

TABLE I: Testes com procura exaustiva dos parâmetros.

| Parâmetros | | | | | Resultados | | |
|---|---|---|---|---|---|---|---|
| **População** | **Gerações** | **Mutações** | **Seleção** | **Crossover** | **Fitness val.** | **Tempo** | **Ger. para convergir** |
| 1000 | 10000 | 0.01 | 3 | 0.5 | 65.2 | 19 min. | Não conv. (4641) |
| 1000 | 10000 | 0.05 | 3 | 0.6 | 60.5 | 19 min. | Não conv. (4253) |
| 1000 | 10000 | 0.05 | 3 | 0.7 | 53.1 | 19 min. | Não conv. (4415) |
| 1000 | 10000 | 0.05 | 3 | 0.8 | 64.99 | 19 min. | Não conv. (3538) |
| 1000 | 10000 | 0.05 | 3 | 0.9 | 48.1 | 19 min. | Não conv. (4036) |
| 1000 | 10000 | 0.01 | 3 | 0.8 | 51.2 | 10 min. | 3570 |
| 1000 | 10000 | 0.01 | 3 | 0.8 | 43.9 | 18:23 min. | 3461 |
| 1000 | 10000 | 0.05 | 2 | 0.8 | 154.8 | 5:37 min. | 1272 |
| 1000 | 10000 | 0.05 | 5 | 0.8 | 53.72 | 19 min. | Não conv. (3972) |

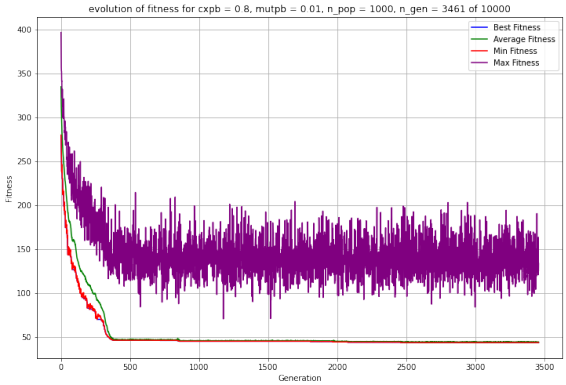


Fig. 4: Evolução do *Fitness* a cada geração do melhor resultado da tabela. 0.01 de mutação, 3 do torneio e 0.8 de *crossover*.

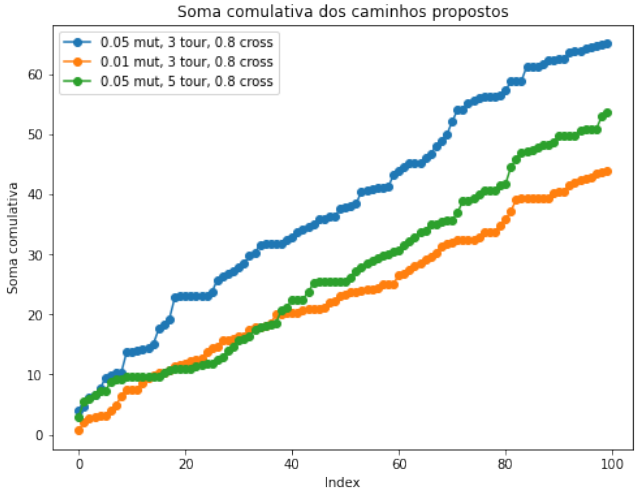


Fig. 5: Evolução da distância total percorrida à medida que o caminho é percorrido, para 3 caminhos dos nossos testes (parâmetros descritos nos labels).

Outra análise que fizemos foi encontrar as sequências em comum entre os diferentes pares de *arrays* distâncias indicam padrões compartilhados entre eles e pode indicar partes dessas soluções que são consistentemente úteis ou eficazes.

O caminho representado a azul e o representado a verde partilham mais sequências em comum do que os outros pares, e essas sequências são também mais longas. Isso pode sugerir uma maior semelhança ou "parentesco" entre esses dois *arrays* em relação ao *array* representado a laranja.

Algumas sequências aparecem em mais de um par de *arrays* (por exemplo, [82, 92, 85]). Isso sugere que essas sequências são especialmente relevantes ou úteis.

Algumas sequências em comum são sub-sequências de outras (por exemplo, [82, 92, 85] é uma sub-sequência de [82, 92, 85, 93]). Isso poderia sugerir que a inclusão de elementos adicionais nessas sequências preserva sua relevância ou utilidade.

**Resultados da tabela:** Taxa de Mutação: Taxas de mutação mais baixas (0.01) parecem resultar numa convergência mais lenta (ou seja, um maior número de gerações para atingir a convergência). Isto é consistente com o objetivo da mutação, que introduz novo material genético e, portanto, ajuda a explorar o espaço de solução. Ao mesmo tempo, o melhor valor de *fitness* não é necessariamente alcançado com uma taxa de mutação mais alta (0.05). Isso sugere que pode haver um equilíbrio a ser encontrado.

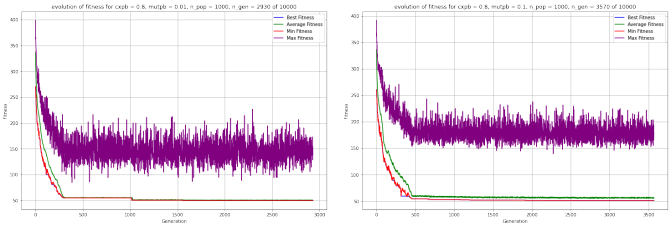

Fig. 6: Ilustração do impacto de variar a mutação (à esquerda *mutation rate*=0.01 e à direita *mutation rate*=0.1)

Taxa de *Crossover*: Há uma tendência geral de diminuição do valor de *fitness* à medida que a taxa de crossover aumenta de 0.5 para 0.9. Isso pode sugerir que uma alta taxa de crossover pode levar a uma convergência prematura, reduzindo a diversidade na população e potencialmente levando a soluções sub-ótimas. Esta observação precisa ser investigada mais a fundo, pois a diferença na convergência também poderia ser devido à aleatoriedade inerente aos algoritmos genéticos.

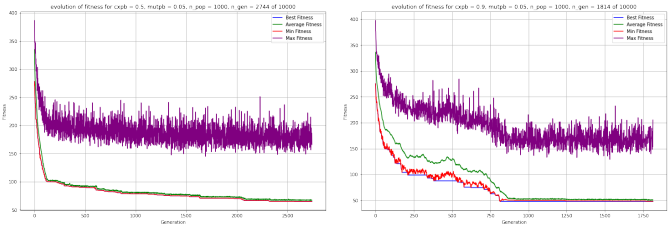

Fig. 7: Ilustração do impacto de variar a mutação (à esquerda *crossover*=0.5 e à direita *crossover*=0.9)

Seleção através do torneio: Os resultados mostram diferentes valores de *fitness* para diferentes tamanhos de torneio. Um tamanho de torneio menor (2) levou a um valor de *fitness* muito maior em comparação com tamanhos de torneio maiores (3 e 5). Isto indica que um tamanho de torneio menor pode levar a melhores soluções neste caso, possivelmente mantendo mais diversidade na população e prevenindo a convergência prematura.

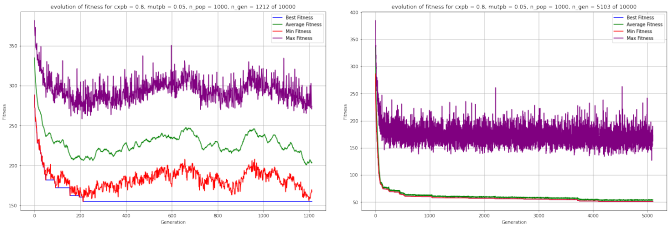

Fig. 8: Ilustração do impacto de variar a mutação (à esquerda *tournament*=2 e à direita *tournament*=5)

Convergência: Em muitos casos, o algoritmo não convergiu dentro do limite de tempo ou geração especificado. Isso pode sugerir que que as configurações de parâmetro não foram ótimas para a convergência absoluta, no entanto são aceitáveis porque provavelmente encontram-se muito perto.

Tempo e Eficiência: O tempo necessário para o algoritmo funcionar variou significativamente, e não parece haver uma relação clara entre o tempo e a qualidade da solução. Isso poderia ser influenciado por vários fatores, incluindo os parâmetros e a aleatoriedade inerente aos algoritmos genéticos.

### A. ***Questão 3.1*** - *Resultados para inputs com tamanhos diferentes*

Foram feitos testes para caminhos de 20, 50 e 80 pontos. Os resultados dependem muitos dos caminhos que foram usados e, obviamente, não tínhamos o caminho ótimo para comparar. O melhor caminho encontrado foi com um *crossover* de 80% de probabilidade de ocorrer, mutação com 1% de probabilidade, população de 10.000 e o número de gerações 10 vezes maior (i.e. 100.000). Os critérios de paragem foram 19 minutos ou se houvesse convergência durante 100 gerações. Foram usados os primeiros *inputs* da tabela que nos foi fornecida, ou seja, por exemplo, quando utilizámos 20 inputs, utilizámos os primeiros 20 da tabela.

TABLE II: Testes com *inputs diferentes*.

| Resultados | | | |
|---|---|---|---|
| **Tam. Input** | **Gerações** | **Fitness** | **Tempo** |
| 20 | 158 | 18.8 | 1:47 min. |
| 50 | 307 | 23.8 | 7:21 min. |
| 80 | 338 | 33.40 | 12:37 min. |
| 100 | 3461 | 43.9 | 18:23 min. |

Com base nos dados fornecidos, parece que existe uma tendência clara de que o tempo de computação, o número de gerações necessárias para a convergência e o valor de fitness aumentam com o aumento do tamanho do input. Isso é esperado, uma vez que problemas de maior dimensão geralmente são mais complexos e, portanto, exigem mais tempo e esforço para serem resolvidos. Em particular, é óbvio que a fitness aumente, dado que esta é a distância total percorrida no caminho, e se o número de ecopontos aumenta, mais temos que ”andar”.

### B. ***Questão 3.2*** - *Melhor caminho*

O melhor caminho encontrado (**para todas os 100 ecopontos**) foi com um *crossover* de 80% de probabilidade de ocorrer, mutação com 1% de probabilidade, população de 100.000 e o número de gerações apesar de estar previsto para 10 vezes o valor da população, após 8h de tempo limite, foi até às 905 gerações. Esta escolha de parâmetros deveu-se aos teste de *setup* previamente efetuados. A sua fitness foi 35.20 e o caminho foi o seguinte: [0, 20, 25, 98, 3, 68, 48, 61, 15, 47, 10, 29, 49, 65, 19, 11, 26, 46, 1, 30, 60, 12, 45, 57, 58, 87, 9, 54, 73, 75, 40, 5, 95, 94, 86, 83, 93, 42, 38, 37, 35, 31, 91, 36, 8, 24, 70, 50, 13, 71, 78, 56, 97, 99, 39, 51, 88, 32, 22, 4, 74, 16, 52, 96, 2, 23, 55, 28, 33, 69, 85, 44, 81, 21, 80, 92, 53, 62, 82, 64, 18, 77, 79, 27, 76, 72, 89, 34, 43, 17, 7, 90, 84, 67, 6, 14, 59, 41, 63, 66, 0].

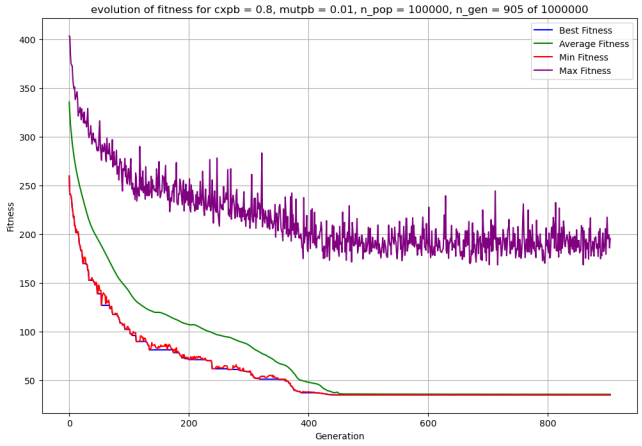


Fig. 9: Evolução dos valores de *fitness* a convergir do melhor caminho que encontrámos (para os 100 ecopontos).

## IV. Mapear as casas e os caminhos

Nesta fase do projeto, já tínhamos essencialmente tudo feito, tínhamos obtido o caminho com menor distância a passar por um determinado número de ecopontos (20, 50, 80, 100). No entanto, apesar de não termos as coordenadas dos ecopontos, mas apenas as distâncias dois a dois, **decidimos tentar mapear as coordenadas de cada ponto**. Assim sendo, **efetuámos uma análise aos dados** (i.e. à matriz de distâncias). Encontrámos alguns pontos que achamos importantes mencionar:

1) A matriz de distâncias não era simétrica (388 dos 4950 pares possíveis de entradas $\left(D_{i,j}, D_{j,i}\right)$ eram diferentes), o que dificulta o mapeamento das distâncias para coordenadas (a matriz deixa de ser uma matriz de distâncias por definição).

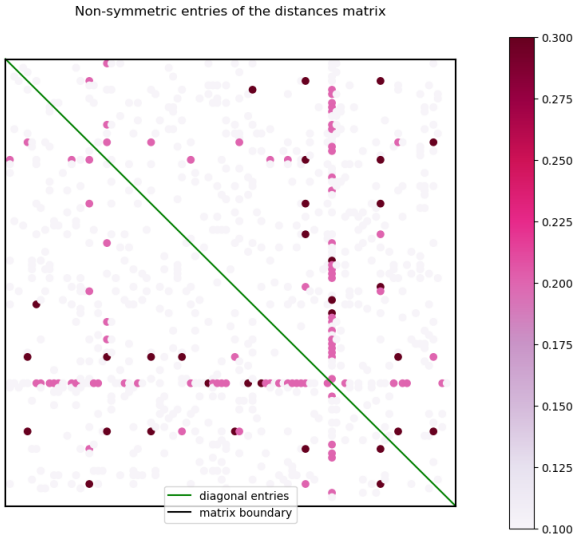


Fig. 10: Ilustração das entradas da matriz que não são simétricas (diagonal a verde).

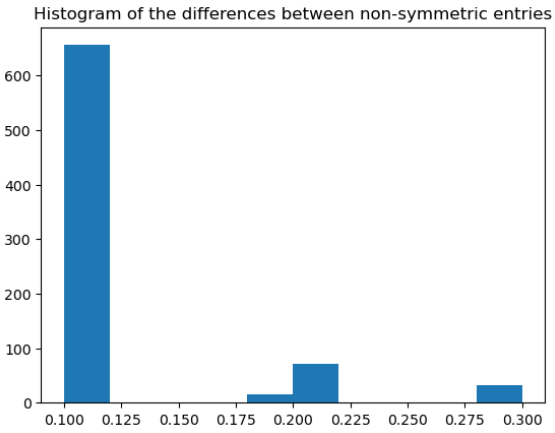


Fig. 11: Histograma das diferenças $|D_{i,j} - D_{j,i}|$ encontradas.

Olhando para os índices onde há este pequeno erro ($\pm$ 0.1, $\pm$ 0.2, $\pm$ 0.3), conseguimos notar que alguns ecopontos em específico verificam mais este erro de simetria (aqueles com linhas quase só com pontos mais escuros na figura 10). De qualquer das formas, a escala do erro é muito baixa.

2) A matriz de distâncias não cumpria a desigualdade triangular. Mais concretamente, (29660 das possíveis 2000000 combinações não cumpriam a desigualdade triangular).

Assim sendo, tornámos a matriz simétrica (tomámos a média de cada par como o novo valor), e aplicámos um método que nos dá as coordenadas aproximadas dos ecopontos (é impossível garantir que são 100% as corretas quando a desigualdade triangular não é cumprida). Para tal, utilizámos o *Multidimensional Scaling Method* (*MDS*). É um método de redução de dimensionalidade. Este recebe a nossa matriz de distâncias (corrigida para ser simétrica) e projeta os pontos num plano 2D. Portanto, não são as verdadeiras coordenadas dos ecopontos, mas o algoritmo tenta manter tão perto quanto possível as distâncias originais. **Isto tudo para podermos visualizar os caminhos que o nosso algoritmo encontrou.** Nota: há que mencionar que as verdadeiras coordenadas poderão ser uma rotação ou reflexão destas.

**Assim sendo, mostrá-los-emos agora:**

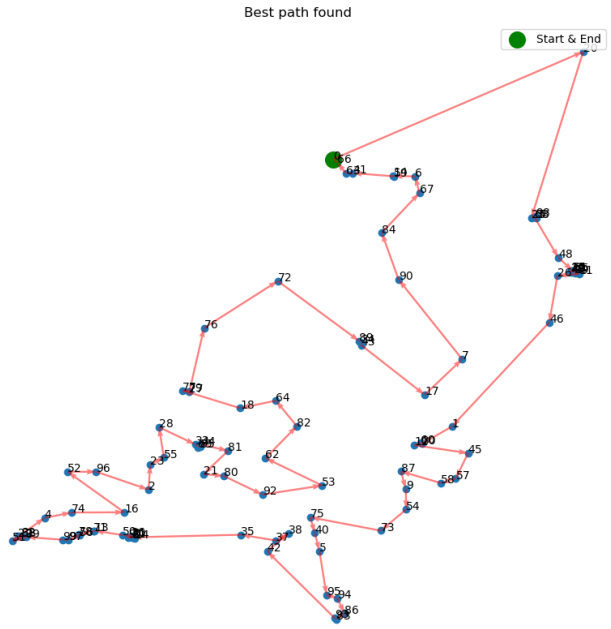


Fig. 12: Visualização do **melhor caminho** encontrado para **100 ecopontos**, com distância 35.2, descrito na secção III-B

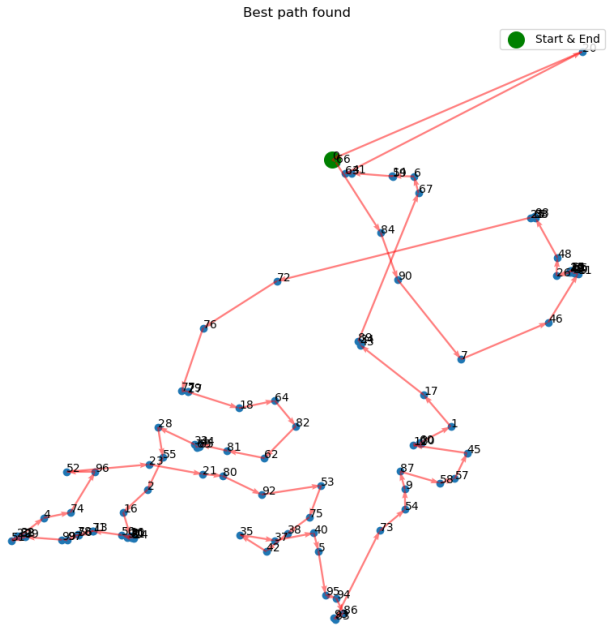


Fig. 13: Visualização do **segundo melhor caminho** encontrado para **100 ecopontos**, com distância 36.8

É muito interessante ver como dois caminhos com "estratégias" tão diferentes acabaram por ter distâncias tão próximas. Podemos também claramente ver que o melhor caminho que encontramos (i.e. o da figura 12) é um excelente (possivelmente o melhor caminho possível, se bem que isso não conseguimos provar).

As próximas figuras mostram os melhores caminhos que encontramos (descritos na tabela II):

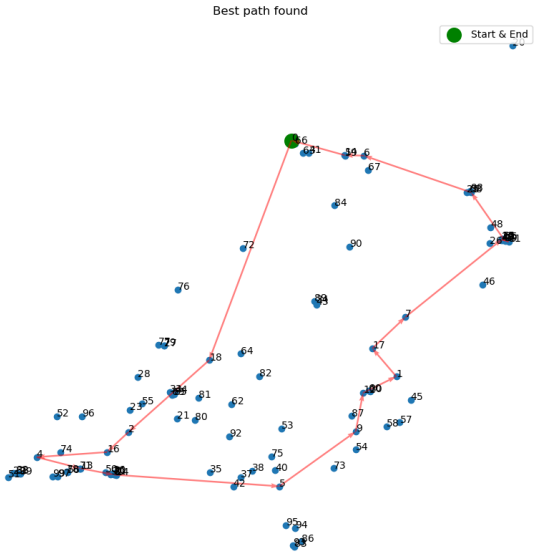


Fig. 14: Visualização do melhor caminho encontrado para **20 ecopontos**, com distância 18.8

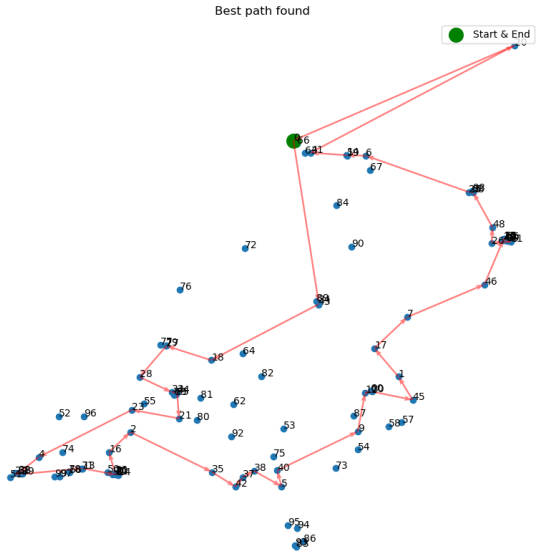


Fig. 15: Visualização do melhor caminho encontrado para **50 ecopontos**, com distância 23.8

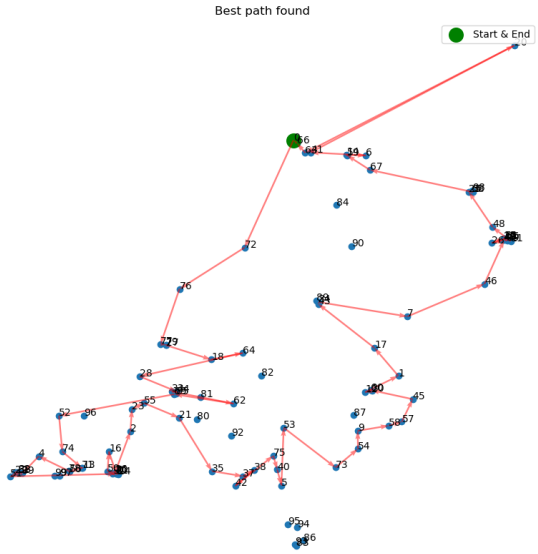


Fig. 16: Visualização do melhor caminho encontrado para **80 ecopontos**, com distância 33.3

## V. Trabalho futuro

Embora os resultados tenham sido bons, ficam algumas sugestões para trabalho futuro:

- Mais variação dos parâmetros e das funções de *crossover*, mutação e seleção, apesar de admitirmos que as escolhas que foram feitas foram ponderadas e muitas das vezes baseadas em conhecimento teórico.
- Implementação de heurísticas adicionais: Além do algoritmo genético, poderíamos ter implementado e comparado o desempenho de outras heurísticas, como a

busca local, *simulated annealing*, algoritmo de colónia de formigas, o da colónia de abelhas, entre outros.

- Melhoramento das condições de paragem depois de atingida a convergência.
- Podíamos ter usado uma *framework* diferente da *DEAP*.

## VI. Conclusão

Após um estudo cuidadoso dos parâmetros que afetam o desempenho do nosso algoritmo genético, estamos em condições de fazer algumas recomendações preliminares.

Nossa análise mostrou que a taxa de mutação e a taxa de *crossover* têm impactos significativos sobre o valor de *fitness* alcançado pelo algoritmo e o número de gerações necessário para atingir a convergência. Taxas de mutação mais baixas, embora resultando em uma convergência mais lenta, não necessariamente levam a melhores valores de *fitness*. Em contraste, taxas de *crossover* mais altas parecem reduzir a diversidade na população e podem levar a soluções sub-ótimas. No entanto, estas observações são preliminares e requerem mais investigação para confirmar.

Observamos também que diferentes tamanhos de torneio resultaram em diferentes valores de *fitness*. Nossos resultados sugerem que um tamanho de torneio menor pode manter mais diversidade na população e evitar a convergência prematura, levando a melhores soluções.

Adicionalmente, nossa análise das sequências em comum entre diferentes pares de *arrays* de distância sugere que existem padrões que são consistentemente úteis ou eficazes. Por exemplo, a sequência [82, 92, 85] aparece em mais de um par de *arrays*, sugerindo que pode ser especialmente relevante. No entanto, como mencionado anteriormente, essas conclusões são observacionais e precisam ser confirmadas por uma análise mais aprofundada.

Finalmente, observamos que o melhor caminho encontrado pelo nosso algoritmo foi alcançado com uma taxa de *crossover* de 80%, uma taxa de mutação de 1%, uma população de 100.000 indivíduos, e 905 gerações. Isso confirma a importância de ajustar adequadamente os parâmetros do algoritmo genético para obter os melhores resultados.

Os algoritmos genéticos são uma classe de algoritmos de otimização e busca heurística inspirados pelos processos naturais de evolução e genética, portanto, apesar de não termos encontrado o melhor caminho possível, foi um bom método para fazer uma boa aproximação tendo em conta os recurso computacionais.

Em suma, nossas descobertas ressaltam a complexidade e a subtileza da otimização de algoritmos genéticos. No entanto, acreditamos que elas proporcionam uma base sólida para futuras investigações e melhorias no desempenho do nosso algoritmo.

## References


[1] Anna Syberfeldt, Patrik Gustavsson, "Robust product sequencing through evolutionary multi-objective optimisation", 2015.

[2] Claudia Canali, Riccardo Lancellotti, "GASP: Genetic Algorithms for Service Placement in Fog Computing Systems", 2019.

[3] https://www.geeksforgeeks.org/tournament-selection-ga/.


## VII. Extra

### A. Algoritmo genético

```python
from deap import base, creator, tools, algorithms
import random
import math
import numpy as np

npop = 300

creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("attr_float", random.uniform, -10, 10)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_float, n=2)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

def eval_func(individual):
    X1, X2 = individual
    Z1 = math.sqrt(X1**2 + X2**2)
    Z2 = math.sqrt((X1 - 1)**2 + (X2 + 1)**2)
    return (math.sin(4 * Z1) / Z1) + (math.sin(2.5 * Z2) / Z2),

toolbox.register("evaluate", eval_func)
toolbox.register("mate", tools.cxBlend, alpha=0.5)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.1)
toolbox.register("select", tools.selTournament, tournsize=3)


pop = toolbox.population(n=npop)
hof = tools.HallOfFame(1)
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("avg", np.mean)
stats.register("std", np.std)
stats.register("min", np.min)
stats.register("max", np.max)

pop, log = algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=10*npop, stats=stats, halloffame=hof, verbose=True)

print(f"Pop: {pop}")
print(f"Log: {log}")
print(f"Hof (X1 e X2): {hof}")

# 3D Plot the function overlapped with the best individual
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')

X1 = np.arange(-10, 10, 0.1)
X2 = np.arange(-10, 10, 0.1)
X1, X2 = np.meshgrid(X1, X2)
Z1 = np.sqrt(X1**2 + X2**2)
Z2 = np.sqrt((X1 - 1)**2 + (X2 + 1)**2)
Z = (np.sin(4 * Z1) / Z1) + (np.sin(2.5 * Z2) / Z2)

ax.plot_surface(X1, X2, Z, cmap='viridis', edgecolor='none', alpha=0.5)
```

```
ax.scatter(hof[0][0], hof[0][1], hof[0].fitness.
    values, c='r', marker='o', s=100)
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_zlabel('Z')
plt.show()

# comparing this value with the best individual
print(f"Best individual: {hof[0].fitness.values}")
```

Listing 1: Algoritmo genético

Os resultados foram X1 = 0.05826170206942979 e X2 = -0.05826170382006746.

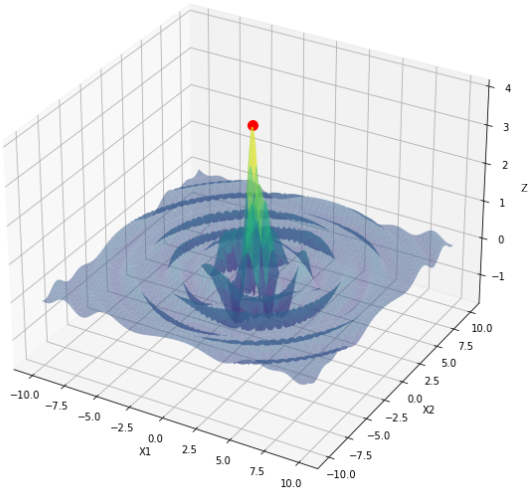


Fig. 17: Visualização 3D do resultados da função f1

### *B. Particle Swarm Optimization*

```
import numpy as np
import pyswarms as ps
from pyswarms.utils.functions import single_obj as
    fx

# Define the function
def f1(X):
    X1, X2 = X[:,0], X[:,1]
    Z1 = np.sqrt(X1**2 + X2**2)
    Z2 = np.sqrt((X1 - 1)**2 + (X2 + 1)**2)
    # minimize bcs PSO is for minimization problems
    return -(np.sin(4 * Z1) / Z1) - (np.sin(2.5 * Z2
    ) / Z2)

# Setup hyperparameters
options = {'c1': 0.5, 'c2': 0.3, 'w':0.9, 'k': 30, '
    p':2}

optimizer = ps.single.GlobalBestPSO(n_particles=100,
     dimensions=2, options=options)

# Optimization
best_cost, best_pos = optimizer.optimize(f1, iters
    =10000)

print(f"Maximum value X1 = {best_pos[0]}, X2 = {
    best_pos[1]}")

```

```
# 3D Plot the function overlapped with the best
    individual
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')

X1 = np.arange(-10, 10, 0.1)
X2 = np.arange(-10, 10, 0.1)
X1, X2 = np.meshgrid(X1, X2)

Z1 = np.sqrt(X1**2 + X2**2)
Z2 = np.sqrt((X1 - 1)**2 + (X2 + 1)**2)
Z = (np.sin(4 * Z1) / Z1) + (np.sin(2.5 * Z2) / Z2)

def z(x1, x2):
    def z1(x1, x2):
        return np.sqrt(x1**2 + x2**2)
    def z2(x1, x2):
        return np.sqrt((x1 - 1)**2 + (x2 + 1)**2)
    return (np.sin(4 * z1(x1, x2)) / z1(x1, x2)) + (
    np.sin(2.5 * z2(x1, x2)) / z2(x1, x2))

ax.plot_surface(X1, X2, Z, cmap='viridis', edgecolor
    ='none', alpha=0.5)
ax.scatter(best_pos[0], best_pos[1], z(best_pos[0],
    best_pos[1]), c='r', marker='o', s=100)
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_zlabel('Z')
plt.show()
```

Listing 2: Particle swarm optimization

Os resultados foram: X1 = 0.058261701805316146 e X2 = -0.058261701777832936.

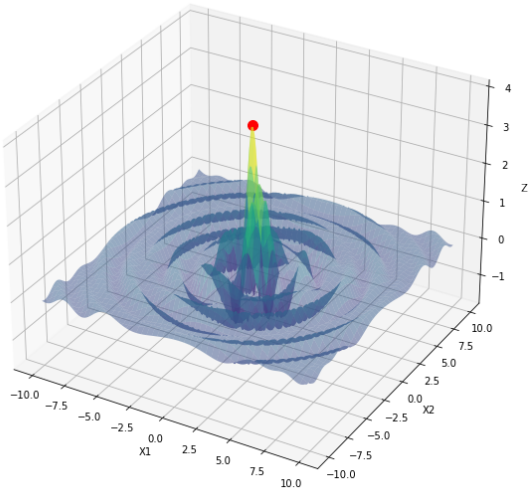


Fig. 18: Visualização 3D do resultados da função f1