Projeto Catch

Bernardo Valente
2171557
Departamento de Engenharia Informática, ESTG, IPLeiria
2171557@my.ipleiria.pt

Tiago Francisco Dias Parente
2170886
Departamento de Engenharia Informática, ESTG, IPLeiria
2170886@my.ipleiria.pt

ABSTRACT

A área da Inteligência Artificial é muito vasta, o que permite resolver inúmeros problemas, que, provavelmente, de outra maneira os seres humanos não seriam capazes de solucionar.

Para a execução da presente experiência foram utilizados algoritmos genéticos. Estes algoritmos foram usados para resolver um problema semelhante ao do caixeiro viajante (TSP), no entanto, em vez de descobrir qual é o melhor caminho entre diferentes cidades, nesta situação pretende-se que sejam apanhadas todas as caixas no menor caminho possível e que o agente saia pela porta, para tal, foram experimentadas várias possibilidades, inspiradas na biologia evolutiva, desde: recombinações (ox, pmx, cx), mutações (insert, swap, inversion, scramble) e vários métodos de seleção (tournament, rank selection). Tudo isto para que sejam atingidos os melhores resultados ao longo das diversas gerações de forma a encontrar uma solução ótima.

1. INTRODUÇÃO

Para a representação do problema foi utilizado um conjunto de células em forma de matriz, em que cada célula tem uma função. No conjunto total das células tem de haver um agente representado com a cor vermelha (na figura abaixo representado por um A), um conjunto de obstáculos representados pela cor cinzenta (na figura abaixo representado por O), um conjunto de caixas representadas pela cor verde (na figura abaixo representado por C) e uma porta representada pela cor preta (na figura abaixo por um P).

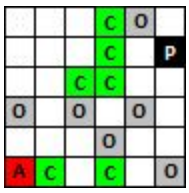

Figura 1 - Representação visual do problema

O objetivo da experiência é fazer com que o agente apanhe todas as caixas evitando os obstáculos e por fim sair pela porta de maneira a que passe pelo caminho mais curto.

Para chegar a este resultado começou-se por calcular as distâncias ótimas do agente a todos os objetos, as distâncias ótimas entre todos os pares de objetos e as distâncias entre todos os pares de objetos e a porta, usando algoritmos de procura no caso do nosso problema usámos o A*.

Para decidir a ordem pela qual as caixas devem ser recolhidas de maneira a minimizar a distância foram usados algoritmos genéticos, que ao longo das gerações vão criando novas populações de indivíduos em que cada indivíduo contém um genoma que é a ordem pela qual as caixas devem ser apanhadas de maneira a que após os métodos de seleção de indivíduos, mutações e recombinações se fique com uma solução que permita apanhar as caixas de maneira mais eficiente.

2. IMPLEMENTAÇÃO

2.1 Descrição do Estado

No projeto para descrever o estado interno utilizou-se uma classe designada CatchState.

Um dos atributos é uma matriz de valores entre 0 e 4 (ver tabela 1) em que cada número tem o seu significado (ver tabela 2).

Tabela 1- Exemplo da representação do estado interno

0 2 0 0 0 4

0 3 3 3 0 0

1 0 0 2 0 0

0 2 0 3 0 3

0 0 0 2 0 0

0 0 0 0 0 0

Tabela 2 -Significado de cada valor

Numero	Significado
0	Celula Vazia
1	Agente
2	Caixa

3	Obstáculo
4	Porta

Dentro da classe há, também, métodos que nos permitem mover o agente de posição tais como o moveUp, moveDown, moveLeft, moveRigth, em que cada um destes métodos é chamado o respectivo canMove de maneira a validar se o agente se pode mover para essa posição, depois da validação se a posição destino for uma caixa o agente apanha essa caixa e decrementa o número de caixas existentes.

Outro método existente dentro da classe CatchState é o computeDistances em que recebe como parâmetro de entrada duas células e o seu objectivo é calcular a distância entre essas células usando a seguinte fórmula

$$|celula1.linha - celula2.linha| + |celula1.coluna - celula2.coluna|$$

no final devolvendo a distancia entre as 2 celulas.

2.2 Descrição do Problema

Para representar o problema usou-se a classe CatchProblemSearch que estende da classe Problem. Dentro desta classe, como atributos, tem-se uma lista que permite guardar as ações disponíveis para o agente realizar, tendo também os atributos que permitem guardar qual é a linha e a coluna objetivo ou seja onde queremos chegar.

Dentro desta classe existem apenas dois métodos:

- O método executeActions que recebe como parâmetro um estado que estenda da classe ou seja a classe CatchState. Este método corre a lista de todas as ações disponíveis, e para cada ação faz a sua validação. Caso seja válida clona o estado e executa a ação para esse estado. No fim adiciona-a a uma lista de estados. No final este método devolve a lista de estados.
- O método isGoal recebe como parâmetro de entrada um estado e valida se o estado passado corresponde ao estado objetivo, caso seja, retorna true, senão retorna false.

2.3 Heurística

A heurística escolhida para a experiência é a distância entre dois pontos. Selecionou-se esta heurística pois adequa-se bem ao que se pretende, que é obter a menor distância possível do agente à porta apanhando todas as caixas.

Para representar a heurística usou-se a classe HeuristicCatch, que estende da classe Heuristic.

Esta classe é relativamente simples, pois o método mais importante que implementa é o método compute em que recebe um estado e apenas chama o método computeDistances (acima descrito) dentro desse estado, retornando no final a distância entre os dois pontos.

2.4 Descrição do Problema para Algoritmos Genéticos

Para representar o problema usado pelo algoritmo genético utilizou-se a classe CatchProblemForGA. Como atributos esta classe tem uma lista de todas as caixas existentes, uma lista de pares (tabela 3), em que estão guardadas todas as posições dos objectos e as distâncias entre si, uma célula para guardar a posição do agente e uma célula para guardar a posição da porta.

A classe tem um método chamado getNewIndividual em que permite criar um novo indivíduo passando a própria classe do problema e o total de número de caixas existentes.

Tabela 3 - Exemplo do conteúdo da lista de pares

2-0 / 0-1: 3
0-1 / 0-5: 4
2-0 / 2-3: 3
2-3 / 0-5: 4
2-0 / 3-1: 2
3-1 / 0-5: 7
2-0 / 4-3: 5
4-3 / 0-5: 6
0-1 / 2-3: 6
0-1 / 3-1: 5
0-1 / 4-3: 8
2-3 / 3-1: 3
2-3 / 4-3: 4
3-1 / 4-3: 3

2.5 Descrição de um Indivíduo

Para representar um indivíduo empregaram-se duas classes dentro do programa: a classe IntVectorIndividual e a CatchIndividual, sendo que a classe CatchIndividual estende da classe IntVectorIndividual.

Quando é criado um indivíduo é lhe atribuído um genoma de forma aleatória que é constituído pela ordem que vai apanhar as caixas, ou seja, o genoma tem de ter o tamanho igual ao número de caixas existentes no problema e cada posição do genoma tem de ter uma posição da caixa a apanhar diferente para ser possível recolher todas as caixas (figura 2).

2	5	0	3	6	1	4	7

Figura 2 - Exemplo de um Genoma

Dentro da classe IntVectorIndividual existe o método swapGenes que recebe um indivíduo e um index e permite trocar o gene na posição do index pelo gene do indivíduo pasado, já dentro da classe CatchIndividual, temos o método computeFitness, que permite avaliar o custo total de uma determinada solução, ou seja, indica qual é o seu fitness.

No início do método iguala-se o fitness com o maior valor positivo possível, pois caso não seja encontrada uma solução retorna um custo infinito. Para calcular o custo de uma solução percorre-se todo o genoma do indivíduo de forma a ver qual é a

ordem das caixas a apanhar, e a cada iteração verifica se a posição do agente à próxima caixa está contida na lista dos pares, se não estiver quer dizer que a solução é impossível, ou seja, retorna um fitness infinito. Caso esteja contida na lista é adicionada a distância entre esse par de objetos a uma variável que guarda a distância total da solução. Depois de percorrer todo o genoma é sinal que o agente já apanhou todas as caixas, ou seja, basta ir à lista de pares ver a distância entre a posição do agente e a porta, e adicionar essa distância ao custo total da distância e atribuir esse mesmo valor ao fitness e retornar o fitness.

Existe, também, o método compareTo que recebe como parâmetro um indivíduo e permite comparar a fitness desse indivíduo com a sua fitness. Caso a sua fitness seja superior à do indivíduo passado retorna 1, caso seja pior retorna -1 e se for igual retorna 0.

2.6 Recombinações

As recombinações permitem criar novos indivíduos com características dos seus pais. Começam por ser selecionados dois indivíduos aleatoriamente, depois são aplicados vários métodos que permitem reproduzir e gerar novos indivíduos com base nas suas características. Para o problema optou-se por usar as recombinações PMX, OX e CX que são largamente utilizadas para resolver o TSP. As recombinações devem ter uma grande probabilidade de ocorrer de maneira a que as novas gerações sejam construídas com base nas anteriores.

2.6.1 Partially-Mapped Crossover (PMX)

O Partially-Mapped Crossover cria descendentes através de valores e ordem herdada dos seus progenitores.

Supondo que se tem os progenitores (12345678) e (37516824), e escolhem-se dois cortes entre a terceira e quarta posição , e entre a sexta e sétima posição, ficando particionados de seguinte forma(123|456|78) e (375|168|24).

A partição central é a partição que vai servir de base para o mapeamento. No exemplo acima descrito tem-se os seguintes mapeamentos, 4<->1, 5<->6 e 6<->8. De seguida a partição central do primeiro progenitor é copiada para o segundo filho, e a partição central do segundo progenitor é copiada para o primeiro filho, ficando com os filhos: filho 1 (***|168|**) e filho 2 (***|456|**). Depois o filho 1 e o filho 2 são "enchidos" com os elementos do correspondente progenitor(progenitor 1 ou 2). No caso de algum elemento já estar presente no filho, este é substituindo usando o mapeamento.

Seguindo o exemplo acima o primeiro elemento do filho 1 seria o valor 1, no entanto este filho já contém esse elemento. Mas usando o mapeamento, o primeiro elemento será então 4 (1<->4). O segundo, terceiro e sétimo elemento podem ser copiados do primeiro progenitor, no entanto o último elemento do filho 1 seria o valor 8, que já existe, logo usando o mapeamento o elemento será o valor 6 (8<->6) e como o valor 6 já está também no genoma faz-se um novo mapeamento para o valor 5(6<->5).

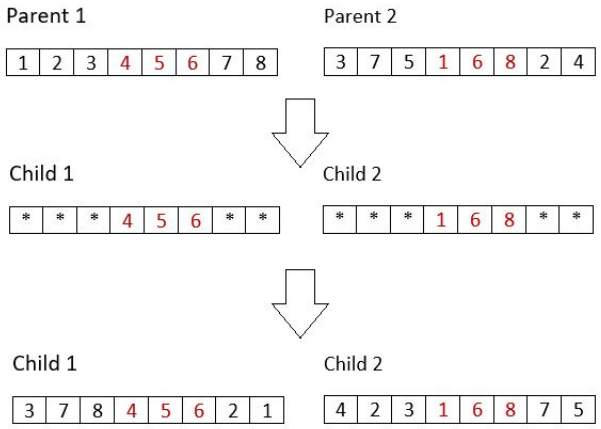

Figura 3 - Exemplo de Recombinação PMX

2.6.2 Ordered Crossover (OX)

O Ordered Crossover cria descendentes escolhendo uma partição de um progenitor, preservando a ordem relativa do outro progenitor.

Supondo que se tem os progenitores (12345678) e (24687531), escolhe-se dois cortes entre a segunda e terceira posição, e entre a quinta e sexta posição, ficando particionados de seguinte forma(12|345|678) e (24|687|531).

Para a criação dos filhos primeiro a partição central de cada progenitor é copiada para os respetivos filhos ficando com os seguintes valores (**|345|***) e (**|687|***). Depois, começando desde o início da terceira partição, dos progenitores inversos ao que foi usado para copiar a partição central, os elementos são copiados pela ordem em que aparecem, omitindo os elementos já existentes. Quando se chega ao fim dos progenitores, começa-se na primeira partição a copiar da mesma forma, resultando nos seguintes filhos:

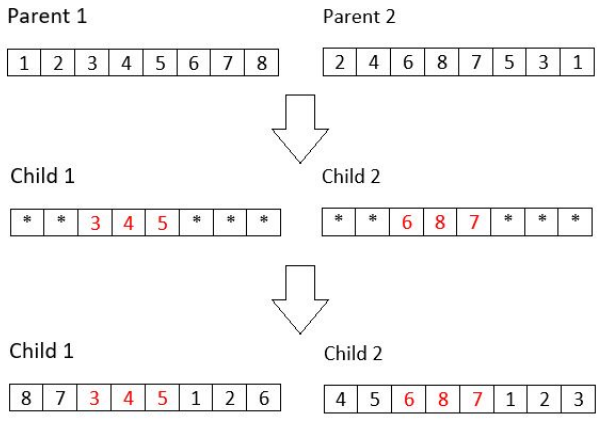

Figura 4 - Exemplo de Recombinação OX

2.6.3 Cycle Crossover(CX)

O Cycle crossover constrói um descendente onde cada posição é ocupada de acordo com um dos seus progenitores.

Supondo que se tem os progenitores(12345678) e (24687531). É escolhido o primeiro elemento para o filho, que pode ser igual ao primeiro elemento progenitor 1 ou progenitor 2. Depois é escolhido o último elemento para esse filho que pode ser o valor 8 ou 1. Como o esse filho já tem o valor 1 no seu genoma é escolhido o valor 8. Neste momento sabe-se que a quarta posição será copiada do primeiro progenitor , pois o valor 8 já se encontra no genoma do filho, e por consequência a segunda posição terá de ser também copiada do primeiro progenitor, pois o valor 4 já está no genoma do filho.

As posições dos elementos escolhidos até este momento são o que se designa por Ciclo.

Para a terceira posição do genoma do filho tanto pode ser herdado do primeiro como do segundo progenitor. Supondo que é selecionado o segundo progenitor, isto implica que a quinta,sexta e sétima posição do filho seja também do segundo progenitor formando assim outro Ciclo.

Podendo concluir que o filho gerado a partir dos valores herdados para cada posição tem em média a mesma quantidade de valores de cada progenitor.

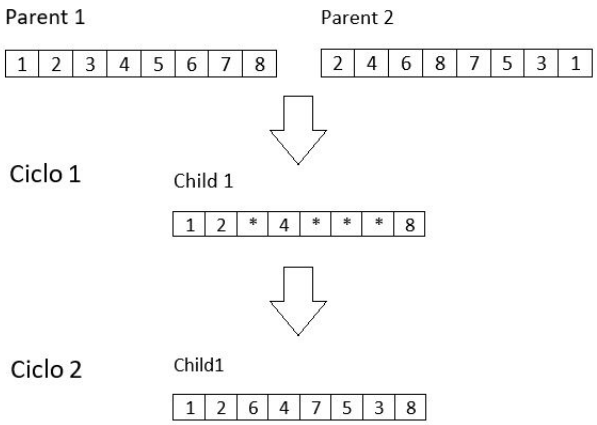

Figura 5 - Exemplo de Recombinação CX

2.7 Mutações

As mutações permitem introduzir dentro de uma população alterações ao genoma o que permite acelerar a procura da solução ideal para um determinado problema, mantendo um nível suficiente de variedade genética na população. No problema optou-se por escolher as mutações Insertion, Swap, Inversion e Scramble. A probabilidade de ocorrer uma mutação,e a sua escolha é definida pelo utilizador.

2.7.1 Insertion

Na operação de Insertion faz-se com que um gene seja escolhido de forma aleatória,que se seja escolhida a nova posição desse gene também de forma aleatória, em que a nova posição tem de ser superior à posição atual. Depois o gene é inserido na nova a posição do genoma fazendo com que os genes entre os dois pontos se movam para trás para que seja possível a inserção.

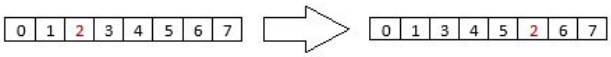

Figura 6 - Exemplo da mutação de inserção

No problema a mutação de Insertion é representada pela classe MutationInsert.

2.7.2 Swap

Na Operação de Swap são escolhidas duas posições aleatórias do genoma e ocorre a troca dos genes selecionados.

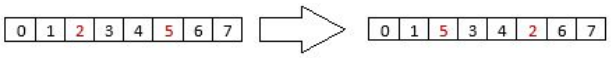

Figura 7 - Exemplo da mutação de troca

No problema a mutação de Swap é representada pela classe MutationSwap.

2.7.3 Inversion

Na operação de Inversion são escolhidas duas posições de forma aleatória dentro do genoma e ocorre a inversão desses genes ficando dispostos de forma invertida dentro do genoma.

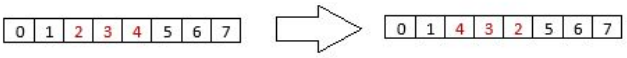

Figura 8 - Exemplo da mutação de inversão

No problema a mutação de Inversion é representada pela classe MutationInversion.

2.7.4 Scramble

Na operação de Scramble são escolhidas duas posições de forma aleatória dentro do genoma e ocorre uma randomização dos genes entre esses dois pontos de maneira a que fiquem dispostos por outra ordem.

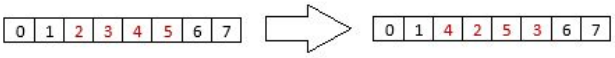

Figura 9 - Exemplo da mutação de randomização

No problema a mutação de Scramble é representada pela classe MutationScramble.

2.8 Métodos de Seleção

Os métodos de seleção permitem selecionar indivíduos (através de um critério) de uma população para que estes sejam passados para as populações futuras de maneira a que ao longo das diversas gerações se encontre a solução ótima.

Neste caso em específico optou-se por usar dois métodos de seleção globalmente utilizados em problemas que envolvem algoritmos genéticos o Torneio(Tournament) e o Rank Selection.

2.8.1 Tournament

Este método de seleção permite escolher um indivíduo a partir de um grupo de indivíduos através de um torneio. Num torneio são selecionados T indivíduos (sendo T o tamanho do torneio) de maneira aleatória onde são comparados, e é escolhido aquele que tiver melhor fitness (figura 10).

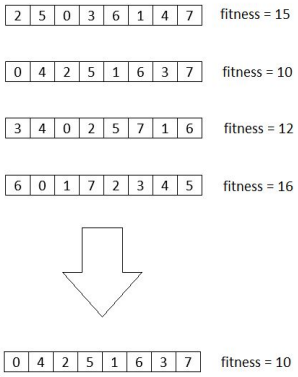

Figura 10 -Exemplo de um torneio de tamanho 4

No programa o Torneio é definido pela classe Tournament.

2.8.2 Rank Selection

Este método de seleção tenta reduzir ao máximo a convergência para um local máximo. A seleção é baseada em rank ordenando os elementos por fitness, ou seja, isto permite que nem sempre os melhores indivíduos sejam escolhidos para as gerações futuras o que faz com que no início não haja um conjunto de indivíduos que seja muito melhor que os outros todos.

Este método guarda numa lista os indivíduos ordenados do pior para o melhor e cada indivíduo tem uma probabilidade de ser escolhido definida por

$$probabilidade = \frac{i}{1+2+...+n}$$

em que o i indica o índice(rank) do indivíduo que se está a calcular a probabilidade. O n indica o número do total de ranks existentes na lista.

No nosso programa a Rank Selection é definida pela classe RankSelection.

2.9 Geração da população inicial

A população inicial condiciona a velocidade e a convergência do algoritmo. Para gerar a população inicial criou-se uma população com n indivíduos especificado pelo utilizador em que cada indivíduo dessa população irá ter um genoma aleatório, ou seja, cada indivíduo pode ter uma ordem diferente para recolher as caixas o que permite ter uma grande aleatoriedade dentro da população.

3. EXPERIÊNCIAS

3.1 Ambiente

Para os testes relativos aos puzzles mais simples (do 1 ao 5) optou-se por testar usando as configurações abaixo descritas, pois devido à sua simplicidade é fácil chegar a uma solução ótima para o problema.

Tabela 4 - Configurações usadas do puzzle 1 ao 5

Tamanho população	50,100
Número de gerações	100,500
Metodo seleção	Torneio tamanho 4
Metodos de recombinação	pmx, ox e cx
Probabilidades de recombinação	0.7,0.8
Métodos de mutação	insert,inversion, swap e scramble
Probabilidades de mutação	0.3, 0.2

Para o puzzle 6 optou-se por fazer testes com mais possibilidades pois já se trata de um problema com maior complexidade e queria-se saber de que maneira poderiam ser obtidos os melhores resultados e quais são as melhores combinações de métodos de seleção, recombinação e mutação para atingir uma solução ótima.

Tabela 5 - Configurações usadas para o puzzle 6

Tamanho população	50, 100, 150
Número de gerações	500, 2000, 5000
Metodo seleção	Torneio tamanho 4,6
Metodos de recombinação	pmx, ox e cx
Probabilidades de recombinação	0.7, 0.8, 0.9
Métodos de mutação	insert,inversion, swap e scramble
Probabilidades de mutação	0.3, 0.2, 0.4

Para o puzzle 7 e 8 usaram-se testes mais específicos de acordo com os dados obtidos dos resultados dos testes para o puzzle 6. Focou-se mais nas possibilidades que permitem obter melhores soluções.

Tabela 6 - Configurações usadas do puzzle 7 e 8

Tamanho população	50,100
Número de gerações	500, 2500
Metodo seleção	Torneio tamanho 4,6
Metodos de recombinação	pmx, ox e cx

Probabilidades de recombinação	0.7,0.8
Métodos de mutação	insert,inversion
Probabilidades de mutação	0.3, 0.2

3.2 Resultados e Discussão

Para analisar os dados dos resultados obtidos fez-se a análise do desvio padrão, que permite verificar o desvio total do melhor fitness para os indivíduos gerados,o que possibilita ver quais são as combinações de valores que permitem obter os melhores resultados.

$$\sigma = fitnessIndividuo - melhorFitness$$

3.2.1 Puzzle 1 ao 5

Devido à simplicidade destes puzzles foi obtido sempre o mesmo valor de fitness para cada puzzle. Qualquer uma das combinações de mutações, seleção e recombinação produziam o mesmo resultado ou seja qualquer combinação de valores permite gerar a solução ideal.

Para o puzzle 1 a melhor fitness encontrada foi 16. Para o puzzle 2 a fitness foi 19, para o puzzle 3 foi 17, para o puzzle 4 foi 15 e para o puzzle 5 foi 45.

Porém, para o puzzle 5 ao analisar-se o ficheiro de fitness médio observa-se uma grande oscilação de valores para o melhor fitness. Sendo que a melhor combinação de valores foi a seguinte:

Tabela 7 - Melhor configuração do puzzle 5

Tamanho população	50
Número de gerações	100
Metodo seleção	Torneio tamanho 4
Metodos de recombinação	pmx
Probabilidades de recombinação	0.7
Métodos de mutação	insert
Probabilidades de mutação	0.3

Ao observar o Gráfico abaixo chega-se à conclusão que para este puzzle a maior parte dos indivíduos gerados tem um desvio padrão entre 3 e 5.

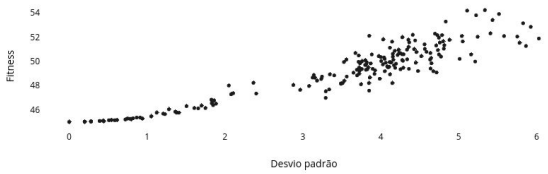

Figura 11 - Desvio padrão e fitness para o puzzle 5

3.2.1 Puzzle 6

Como descrito no ambiente para o puzzle 6 optou-se por realizar mais testes de maneira a ter mais dados para analisar e saber quais são as melhores combinações para resolver o nosso problema .

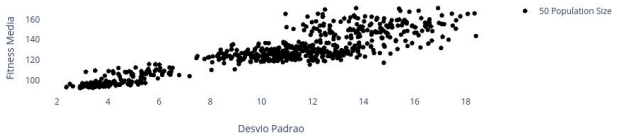

Figura 12 - Desvio padrão e fitness no puzzle 6 para população 50

Ao analisar este gráfico constata-se que os indivíduos gerados com população de 50 indivíduos têm mais tendência para terem um desvio padrão entre 8 e 12 porém também há uma grande quantidade de indivíduos com o desvio padrão entre 2 e 6.

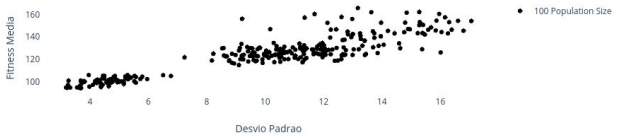

Figura 13 - Desvio padrão e fitness no puzzle 6 para população 100

Ao analisar este gráfico verifica-se que os resultados obtidos para a população 100 são semelhantes à população 50 em termos dos indivíduos gerados na zona do desvio padrão entre 8 e 12, porém com população 100 há também uma grande concentração de indivíduos a ser gerados com valores de desvio padrão entre 4 e 6.

Analisando o ficheiro dos melhores indivíduos conclui-se que a população que permite obter os melhores resultados é a população de tamanho 50.

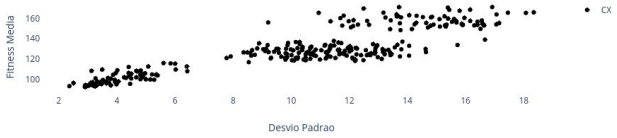

Figura 14 - Desvio padrão e fitness no puzzle 6 para o método de recombinação CX

Figura 15 - Desvio padrão e fitness no puzzle 6 para o método de recombinação PMX

Observando estes dois gráficos pode-se ver que os indivíduos gerados com a recombinação pmx e cx tem uma grande tendência para terem um desvio padrão entre 8 e 12 , havendo também uma grande quantidade de indivíduos com o desvio padrão entre 2 e 6.

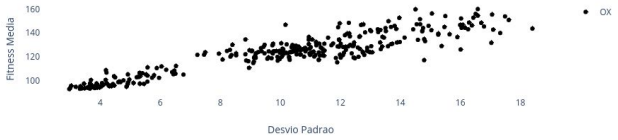

Figura 16 - Desvio padrão e fitness no puzzle 6 para o método de recombinação OX

Analisando este gráfico verifica-se que os indivíduos gerados têm tendência para ter um desvio padrão entre 8 e 12, porém também há um grande conjunto de indivíduos gerados com um desvio padrão entre 3 e 4.

Ao comparar todas as recombinações analisadas chega-se à conclusão que todos os métodos de recombinação acabam por gerar indivíduos parecidos entre si, sem que haja grandes diferenças porém o pmx e o cx permitem gerar indivíduos com menor desvio padrão.

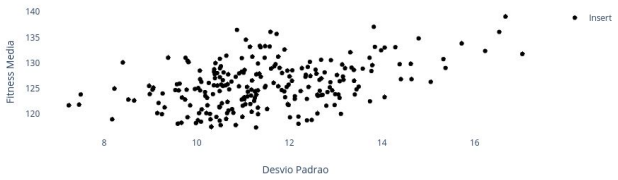

Figura 17 - Desvio padrão e fitness no puzzle 6 para o método de mutação Insert

Figura 18 - Desvio padrão e fitness no puzzle 6 para o método de mutação Swap

Figura 19 - Desvio padrão e fitness no puzzle 6 para o método de mutação Scramble

Analisando os gráficos acima constata-se que indivíduos gerados têm valores muitos dispersos de fitness, ou seja, não há um padrão de comportamento para estes indivíduos.

Figura 20 - Desvio padrão e fitness no puzzle 6 para o método de mutação Inversion

Analisando este gráfico verifica-se que os indivíduos gerados usando a mutação de Inversion tem tendência para estar todos dentro de um desvio padrão entre 2 e 6, o que indica que há uma grande consistência de indivíduos gerados.

Comparando todos os métodos de mutação vê-se que o Inversion é o que gera os melhores indivíduos pois permite mais consistência na sua geração,tendo também um desvio padrão baixo.

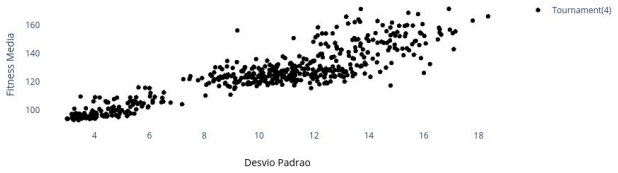

Figura 21 - Desvio padrão e fitness no puzzle 6 para o método de seleção Torneio de quatro

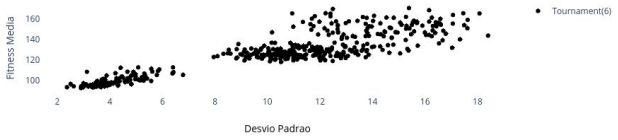

Figura 22 - Desvio padrão e fitness no puzzle 6 para o método de seleção Torneio de Seis

Ao analisar estes dois gráficos conclui-se que os indivíduos gerados quer para o torneio de quatro quer para o torneio de seis são muito parecidos, em que a maior parte dos indivíduos são gerados com um desvio padrão entre 8 e 12, porém também entre 3 e 6. Contudo, analisando o ficheiro dos melhores indivíduos chega-se à conclusão que o torneio que permite obter os melhores resultados é o de quatro.

Figura 23 - Desvio padrão e fitness no puzzle 6 para a probabilidade de mutação de 0.2

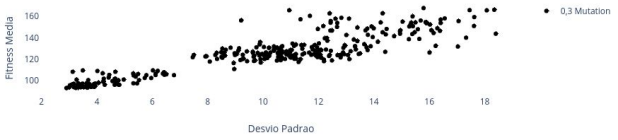

Figura 24 - Desvio padrão e fitness no puzzle 6 para a probabilidade de mutação de 0.3

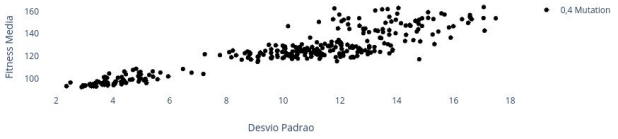

Figura 25 - Desvio padrão e fitness no puzzle 6 para a probabilidade de mutação de 0.4

Analisando estes três gráficos verifica-se que os indivíduos por eles gerados são muito parecidos entre si não havendo grandes diferenças nos gráficos. Vê-se que os indivíduos gerados estão entre dois conjuntos de desvios padrões 2-4 e 10-12.

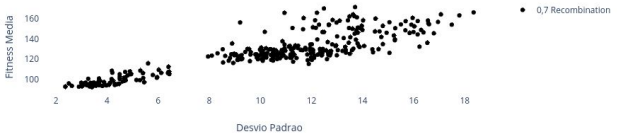

Figura 26 - Desvio padrão e fitness no puzzle 6 para a probabilidade de recombinação de 0.7

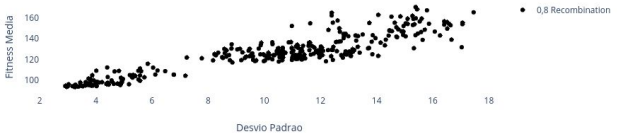

Figura 27 - Desvio padrão e fitness no puzzle 6 para a probabilidade de recombinação de 0.8

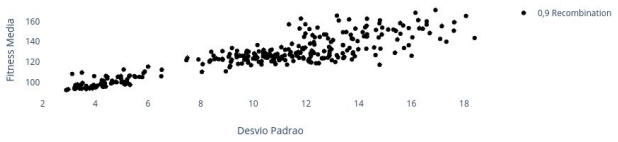

Figura 28 - Desvio padrão e fitness no puzzle 6 para a probabilidade de recombinação de 0.9

Tal como para as probabilidades de mutação os gráficos das diversas probabilidades de recombinação acabam por ser muito parecidos entre si,gerando indivíduos muito semelhantes entre as diferentes recombinações.

Depois de analisados todos os gráficos verifica-se que para obter os melhores valores podemos usar qualquer uma das configurações da tabela abaixo.

Tabela 8 - Características para obter o melhor indivíduo

Tamanho população	50
Número de gerações	500
Metodo seleção	Torneio tamanho 4
Metodos de recombinação	pmx, ox e cx
Probabilidades de recombinação	0.7,0.8
Métodos de mutação	insert,inversion,swap ,scramble
Probabilidades de mutação	0.3, 0.2,0.4

Para o puzzle 6 o melhor fitness obtido foi 89.

3.2.1 Puzzle 7

Como era de prever, os resultados obtidos dos testes para o puzzle 7 estão semelhantes aos observados pelo puzzle 6.

Como explicado na definição do ambiente focou-se em testes mais restritos em que foram testadas menos possibilidades de combinações devido à semelhança com os testes discutidos anteriormente no puzzle 6.

O que diferencia os melhores indivíduos deste puzzle para o anterior é a obtenção de um indivíduo com um fitness ótimo usando um torneio de seis.

Tabela 9 - Características para obter o melhor indivíduo

Tamanho população	50
Número de gerações	500
Metodo seleção	Torneio tamanho 4,6
Metodos de recombinação	pmx, ox e cx
Probabilidades de recombinação	0.7,0.8
Métodos de mutação	insert,inversion
Probabilidades de mutação	0.3, 0.2

Para o puzzle 7 o melhor fitness obtido foi 86.

3.2.1 Puzzle 8

Como era de esperar, os resultados obtidos dos testes para o puzzle 8 estão semelhantes aos observados pelo puzzle 6.

Relativamente aos indivíduos gerados no puzzle 8 verifica-se que geração dos mesmos está de acordo com o observado no puzzle 6.

Observando o ficheiro dos melhores indivíduos constata-se que o melhor indivíduo foi obtido apenas com a recombinação pmx chegando-se, assim, à conclusão que esta recombinação é ideal quando temos um grande número de caixas a serem recolhidas.

Tabela 10 - Características para obter o melhor indivíduo

Tamanho população	50
Número de gerações	500
Metodo seleção	Torneio tamanho 4
Metodos de recombinação	pmx
Probabilidades de recombinação	0.7,0.8
Métodos de mutação	insert,inversion
Probabilidades de mutação	0.3, 0.2

Para o puzzle 8 o melhor fitness obtido foi 100.

4. EXTRAS

Devido às limitações existentes nos recursos computacionais foi necessário criar uma aplicação de consola, multiplataforma, de forma a ser possível adquirir um servidor na cloud para correr os testes. Esta aplicação tem a mesma lógica de negócio que a aplicação GUI, todavia, apenas recebe como parâmetro de entrada o nome da configuração para a qual vai correr os testes. Para cada puzzle é criada uma pasta individual onde vão ser guardados os dados resultantes dos testes para ser possível uma análise mais fácil. O nome das configs tem de acabar com um número entre 0 e 8 de acordo com o puzzle para o qual vamos correr os testes,sendo obrigatório possuir a extensão .txt.

Para correr os testes basta abrir uma consola , alterar o ficheiro script.sh(unix) ou script.bat(windows) adicionando as configs que queremos correr e em seguida basta correr o script.

5. CONCLUSÃO

Os resultados obtidos foram ao encontro do pretendido, permitindo reunir e retirar bastante informação para a compreensão dos dados, facilitando, assim, saber quais os melhores valores a serem usados para obter as melhores soluções para cada puzzle.

Para obter estes resultados corremos uma vasta coleção de testes, sendo que para os primeiros cinco puzzles devido à sua simplicidade optámos por testes mais simples, pois com qualquer combinação de valores de recombinação, mutação e seleção chegámos sempre à melhor solução. Para o puzzle 6 corremos um teste com mais valores e mais possibilidades de combinações para perceber de que maneira conseguiríamos obter os melhores indivíduos. Para o puzzle 7 e 8 tendo já conhecimento dos resultados obtidos anteriormente optámos por fazer um teste mais restrito de maneira a focarmo-nos em obter a melhor solução possível. Chegámos à conclusão que todas as recombinações são muito parecidas entre si ou seja, geram indivíduos muito semelhantes. As mutações permitem com que cheguemos à solução ótima mais rápido. Para todas as mutações implementadas obtivemos o melhor valor de fitness ou seja a obtenção do melhor indivíduo depende dos métodos de recombinação, porém, a mutação de Inversion permite a geração de indivíduos mais consistentes.O tamanho da população ideal para os indivíduos é de 50 e o melhor método de seleção é o torneio com tamanho quatro. Relativamente às probabilidades de recombinação usámos valores altos entre 0.7-0.9 pois, permite gerar indivíduos de acordo com os seus progenitores, fazendo com que seja possível ao longo das gerações gerar cada vez melhores indivíduos. Relativamente às probabilidades de Mutação usamos valores baixos contidos entre 0.2-0.4 pois, permite introduzir novo material genético numa população sem que se perca as parecenças com os seus progenitores.

Quando temos um puzzle com muitas caixas observámos que a recombinação PMX permite obter melhores resultados quando comparado às recombinações OX e CX.

Começámos pela perceção do problema e das classes fornecidas pelos docentes, servindo de ponto de partida para a realização do projeto. Passada a fase da perceção começou a fase de implementação em que fizemos uma grande pesquisa de maneira a ultrapassar os problemas que iam surgindo para que no final ficássemos satisfeitos com o resultado final,adquirindo uma grande quantidade de conceitos pelo caminho. Terminada a fase de implementação começou a fase dos testes em que devido à falta de poder de processamento levou à criação de uma nova aplicação para correr os testes num servidor na cloud. Este processo levou a que aprendêssemos novos conceitos relativos a cloud computing. Depois dos testes terminados passámos à fase de análise de dados em que desenvolvemos a nossa capacidade de retirar informações e conclusões através da criação de gráficos e interpretação de ficheiros.

Concluindo, consideramos ter atingido os objetivos do projeto, contribuindo não só para alargar os nossos conhecimentos relativos à área da Inteligência Artificial mas também outras áreas da Informática.

6. AGRADECIMENTOS

Agradecemos ao docente José Ribeiro por toda a ajuda e tempo disponibilizado para o esclarecimento das dúvidas que iam surgindo, e por nos orientar de maneira a chegar aos melhores resultados e por consequência conseguirmos chegar ao nosso objectivo.

7. BIBLIOGRAFIA

[1] GitHub (2015). GeneticAlgorithm. Consultado a 16/05/2019 Disponível em https://github.com/nsadawi/GeneticAlgorithm/blob/master/src/Mutation.java

[2] GitHub (2012). Solving-the-TSP-using-Genetic-Algorithms.. Consultado a 17/05/2019 Disponível em https://github.com/PLT875/Solving-the-TSP-using-Genetic-Algorithms/tree/master/src/Crossover

[3] Research Gate (2015). Crossover Operators in genetic algorithms: a review. Consultado a … . Disponível em https://www.researchgate.net/publication/288749263_CROSSOVER_OPERATORS_IN_GENETIC_ALGORITHMS_A_REVIEW

[4] Mnemosyne_studio() (S.D.). Mutation. Consultado a 25/05/2019 Disponível em http://mnemstudio.org/genetic-algorithms-mutation.htm

[5] Mnemosyne_studio() (S.D.). Selection and Fitness. Consultado a 25/05/2019 . Disponível em http://mnemstudio.org/genetic-algorithms-selection.htm

[6] Ahmad B. A. Hassanat *, Esra'a Alkafaween (S.D.). On Enhancing Genetic Algorithms Using New Crossovers. Consultado a 18/06/2019 . Disponível em https://arxiv.org/ftp/arxiv/papers/1801/1801.02335.pdf

[7] ABDOUN Otman, ABOUCHABAKA Jaafar (2011). A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem. Consultado a 18/06/2019 . Disponível em https://arxiv.org/ftp/arxiv/papers/1203/1203.3097.pdf