FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Licenciatura em Engenharia Informática

Introdução à Inteligência Artificial
2018/2019 - 2º Semestre

Trabalho Prático Nº2
DEIBaLLZ
The Evolutionary Tatic

Trabalho realizado por:

Ana Carolina Ribeiro Bandeira	2016222843	abandeira@student.dei.uc.pt
Francisco Miguel Almeida Monteiro	2016241480	fmonteiro@student.dei.uc.pt
Ricardo José Monteiro Paiva	2016253100	rjpaiva@student.dei.uc.pt

Introdução:

Para este trabalho, foi-nos pedido para estudar e implementar um controlador baseado em um algoritmo genético. Um algoritmo genético é um algoritmo evolutivo, usado em problemas de busca para obter uma solução aproximada. Esta solução
é utilizada para guiar um carro no jogo “DEIBallZ” onde o objetivo é inserir a bola dentro da baliza adversária.
O presente relatório tem como objetivo descrever todas as opções tomadas na construção da solução para o problema proposto bem como a análise dos testes executados com os diferentes algoritmos e funções.

Modelização:

Para este trabalho começámos por modelar o nosso problema. Desta forma, procurámos responder às seguintes questões:

- O que é uma população?
  Uma população corresponde a uma lista de indivíduos gerados de uma forma aleatória. Esta população vai ser posteriormente utilizada para calcular o indivíduo mais vantajoso, ou seja, com melhor aptidão para a obtenção da solução pretendida.

- O que é e para que serve a função de fitness?
  A função de fitness vai aceder a cada um dos indivíduos da população e vai-lhes atribuir um valor. Este valor é calculado em função de vários parâmetros que vão ser discutidos mais à frente neste relatório. Geralmente os indivíduos que obtêm um valor alto na função de fitness, são os indivíduos mais aptos e com uma maior probabilidade em serem escolhidos.

- O que é o mecanismo de seleção por torneio?
  Corresponde a um método de seleção de um indivíduo de uma população em função do seu valor de fitness. Esta seleção escolhe N indivíduos de uma população, e simula um pequeno torneio onde os indivíduos selecionados “competem” entre si até obtermos o melhor indivíduo desse grupo.

- O que é o Crossover?
  O crossover, ou recombinação, é um operador genético para variar o conteúdo dos genes de um indivíduo. Esta operação pega em dois indivíduos dentro de uma população, selecionados previamente, e através da sua combinação gera um novo indivíduo com características herdadas dos seus parentes.

- O que é a Mutação?
  É um operador genético que pega no conteúdo de um genótipo e altera o conteúdo dos seus genes, tendo em conta uma probabilidade de mutação.

- O que é Substituição?

  Seguindo o modelo elitista, os melhores indivíduos passam inalterados para a geração seguinte.

Primeira meta - Desenvolvimento:

Para a primeira meta, foi-nos pedido para implementar o algoritmo genético e fixar a topologia da rede neuronal. Desta forma, decidimos usar o exemplo fornecido no enunciado e definir a arquitetura da seguinte maneira:

- Camada de entrada: 15 nós;
- Camada Escondida: 20 nós;
- Camada de Saída: 2 nós;

Para a construção do algoritmo genético, recorremos ao pseudo-código presente na bibliografia da cadeira. Em primeiro lugar foi implementado o método que inicializa a população:

```
public override void InitPopulation() {
population = new List<Individual>();
while (population.Count < populationSize)
{
    GeneticIndividual new_ind = new GeneticIndividual(topology);
    new_ind.Initialize();
    population.Add(new_ind);
}
}
```

Este método, cria novos indivíduos inicializando-os com valores gerados aleatoriamente e adiciona-os à população.
Depois, é feita a seleção por torneio de dois indivíduos, que escolhe aqueles com melhor aptidão. Estes indivíduos podem ser submetidos a crossover que corresponde à troca do genótipo entre os pais. A probabilidade de estes sofrerem crossover é determinada pela geração de um número aleatório que, ao ser posteriormente comparado com a constante de crossover passada como parâmetro, determina se estes indivíduos se podem reproduzir ou não. De uma forma semelhante, a mutação também só é executada caso, a probabilidade de esta ocorrer se encontre de acordo com o valor aleatório gerado.
Após ocorrerem estas transformações, os indivíduos gerados são adicionados a uma lista que corresponde à nova população que vai ser actualizada na próxima geração.
Para testar várias perspectivas, foram geradas diferentes funções de fitness.

O que é uma boa Função de Fitness?

A função de fitness calcula a pontuação de cada indivíduo. Quanto maior for a pontuação e quanto mais casos de boa pontuação houverem, melhor é a função de Fitness. As novas gerações são criadas com o objetivo de apresentarem melhor aptidão, sendo que os progenitores utilizados para a reprodução e cruzamento de genes são escolhidos com base na sua aptidão, ou seja, consoante o melhor fitness.

Função de fitness #1:

```
public float GetScore()
{
    int difGolos;
    float fitness;
    difGolos = GoalsOnAdversaryGoal - GoalsOnMyGoal;

    //Diferença de golos
    // Se for menor que 0
    if (difGolos < 0)
    {
        // Vamos dar um valor de aptidão baixo para este elemento
        fitness = difGolos;

    }
    else
    {
        // Caso de empate
        if (difGolos == 0)
        {
            fitness = 10;
        }
        // Caso esteja com vantagem
        fitness = difGolos * 100 * Math.Abs(avgSpeed);

    }
    return fitness;
}
```

Esta função verifica a diferença de golos marcados, e devolve esse valor a multiplicar pelo absoluto da velocidade média e por um fator inteiro que favorece os golos (100), caso a diferença de golos seja positiva. Em caso de empate, devolve 10. Caso a diferença de golos seja negativa, devolve essa mesma diferença.

Função de fitness #2:

```
public float GetScore()
{
    int difGolos;
    float fitness;
    difGolos =  GoalsOnAdversaryGoal -  GoalsOnMyGoal;

    //Diferença de golos
    // se estiver empatado
    if (difGolos == 0)
    {
        fitness = 1 / distancefromBallToAdversaryGoal;
        fitness = (float)Math.Pow(fitness,3);
        return fitness;
    }
    else
    {
        fitness = difGolos*2;
        fitness = (float)Math.Pow(fitness, 3)*driveTime;
        return fitness;
    }
}
```

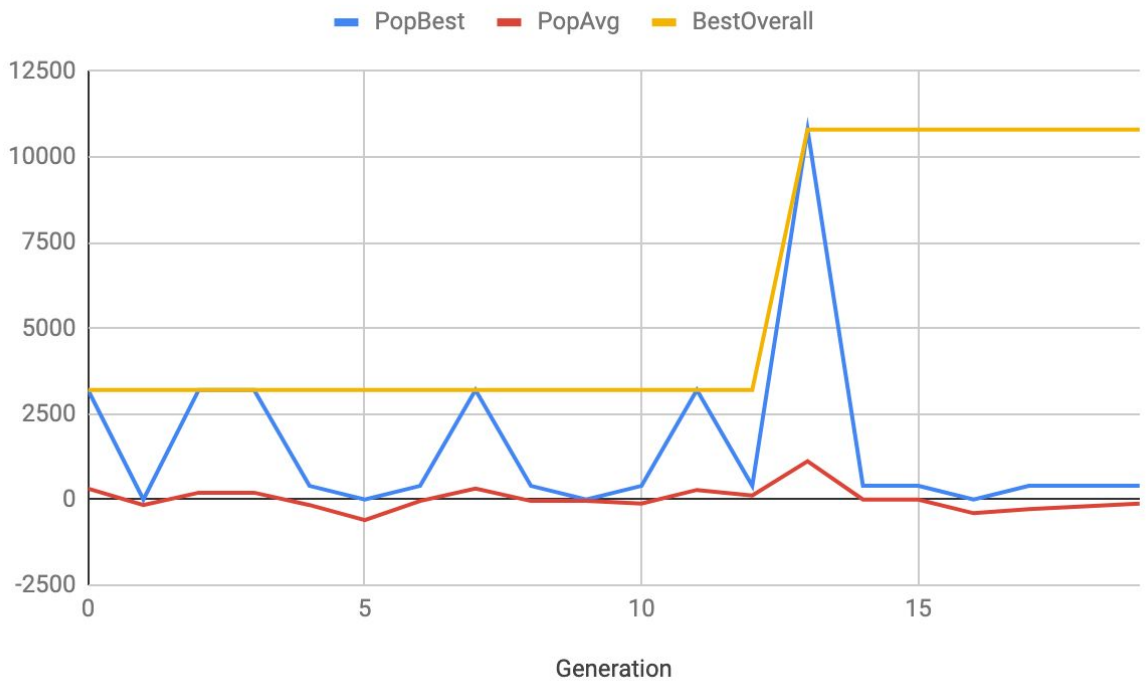
Esta função de fitness, é semelhante à anterior. No entanto, em caso de empate tenta minimizar a distância total a bola à baliza adversária. Pega neste valor e eleva-o ao cubo. Caso não seja empate devolve o valor da diferença de golos elevado ao cubo, e multiplicado pelo driveTime.
A razão de se usar a função exponencial é de forma a fazer com que os valores da função de fitness se afastem entre si. Usámos a elevação ao cubo, de forma a preservar o sinal da diferença de golos.

Segunda meta - Experimentação e análise:

Esta meta tinha como objetivo a análise da eficácia e robustez do algoritmo implementado em diferentes cenários possíveis. Para isso, o algoritmo foi testado várias vezes com diferentes valores passados como parâmetros como se pode ver em baixo:
Nota: Ao longo destes testes as seeds aleatórias foram mudadas regularmente

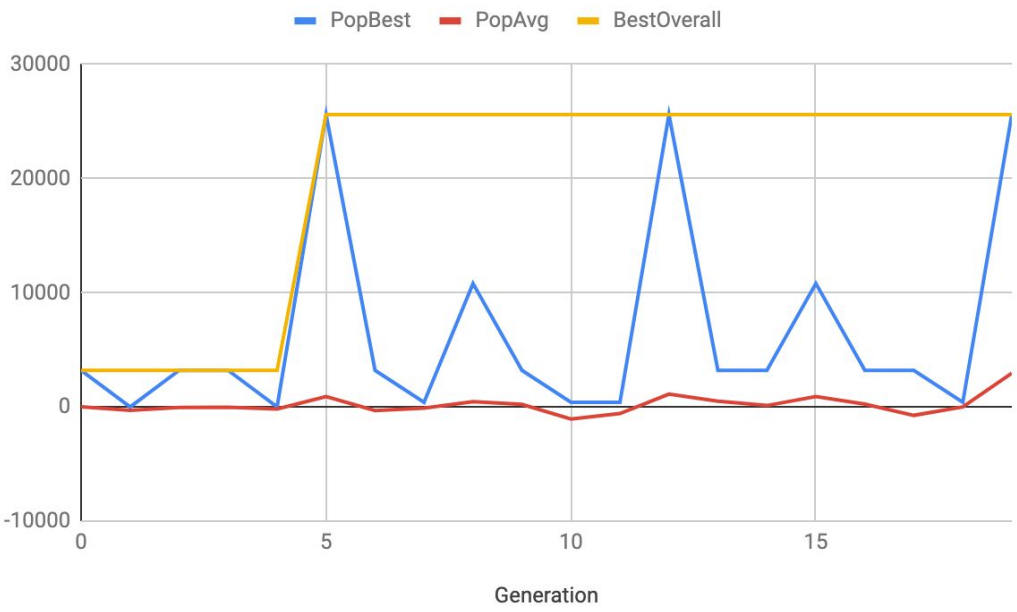
Caso de teste #1:

Population Size: 10
Num Generations: 20
Mutation Probability: 0.01
Crossover Probability: 0.01
Tournament Size: 5
Elitismo: No



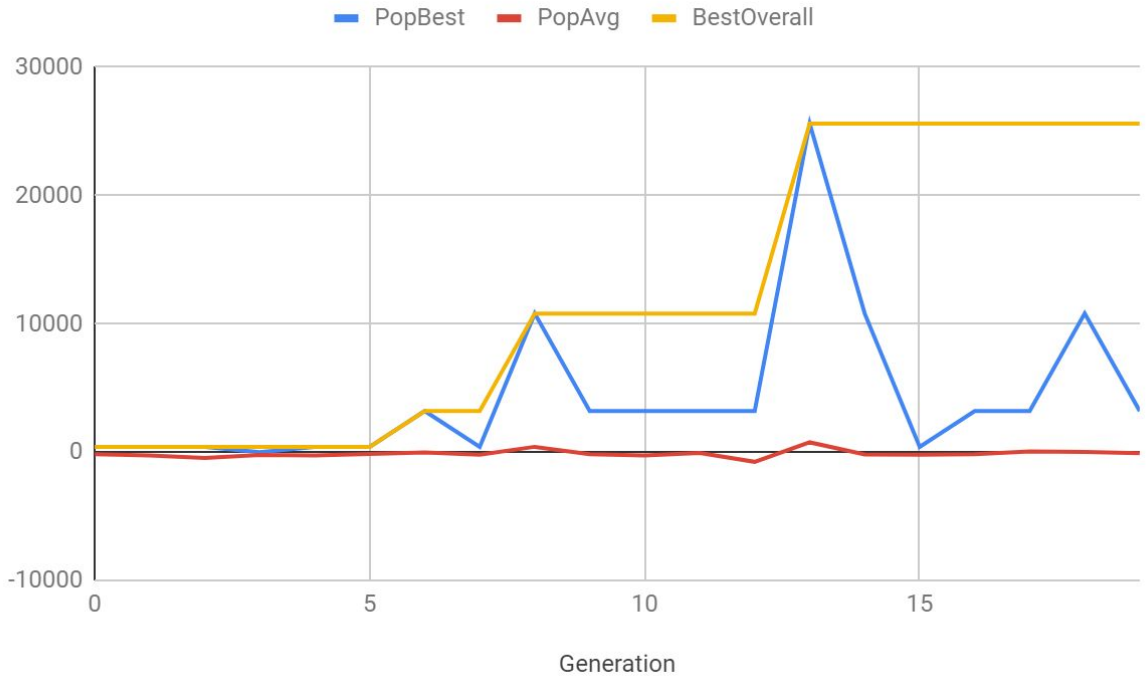
Caso de teste #2:

Population Size: 20
Num Generations: 20
Mutation Probability: 0.01
Crossover Probability: 0.01
Tournament Size: 5
Elitismo: No



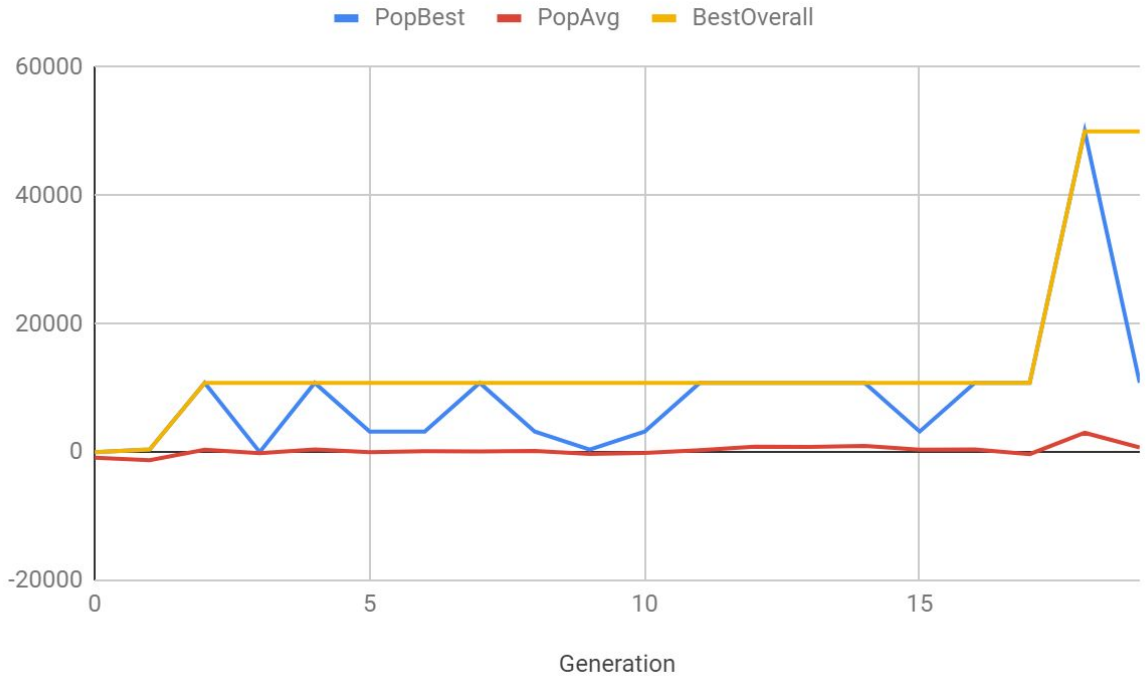
Caso de teste #3:

Population Size: 40
Num Generations: 20
Mutation Probability: 0.01
Crossover Probability: 0.01
Tournament Size: 5
Elitismo: No



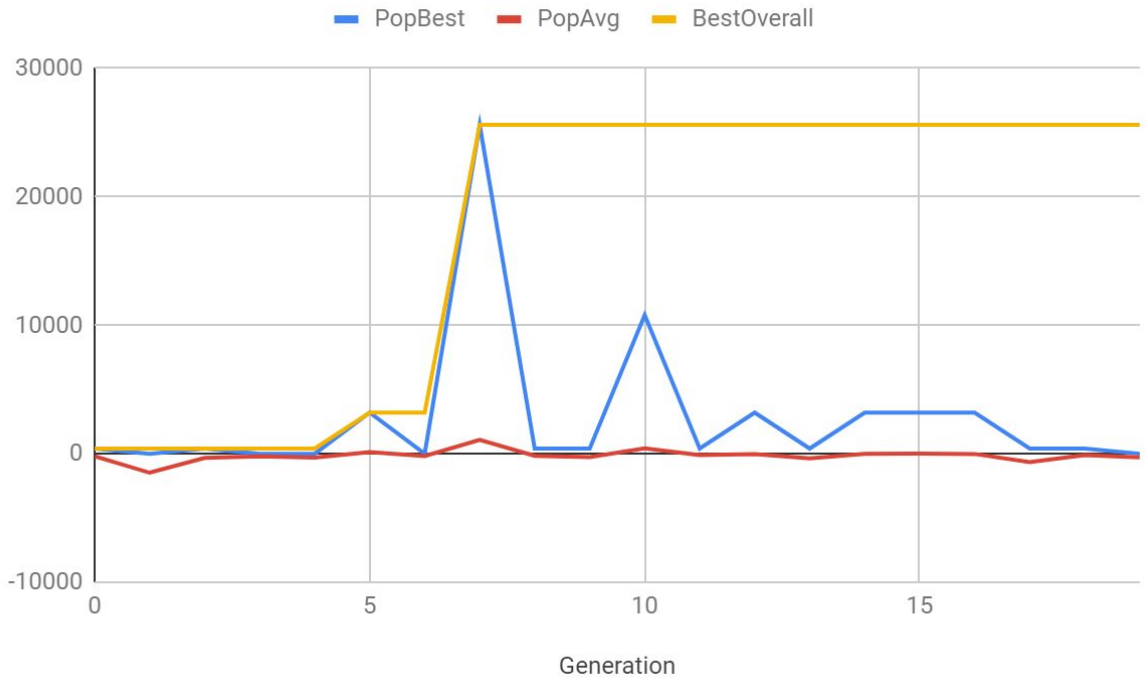
Caso de teste #4:

Population Size: 20
Num Generations: 20
Mutation Probability: 0.05
Crossover Probability: 0.01
Tournament Size: 5
Elitist: No



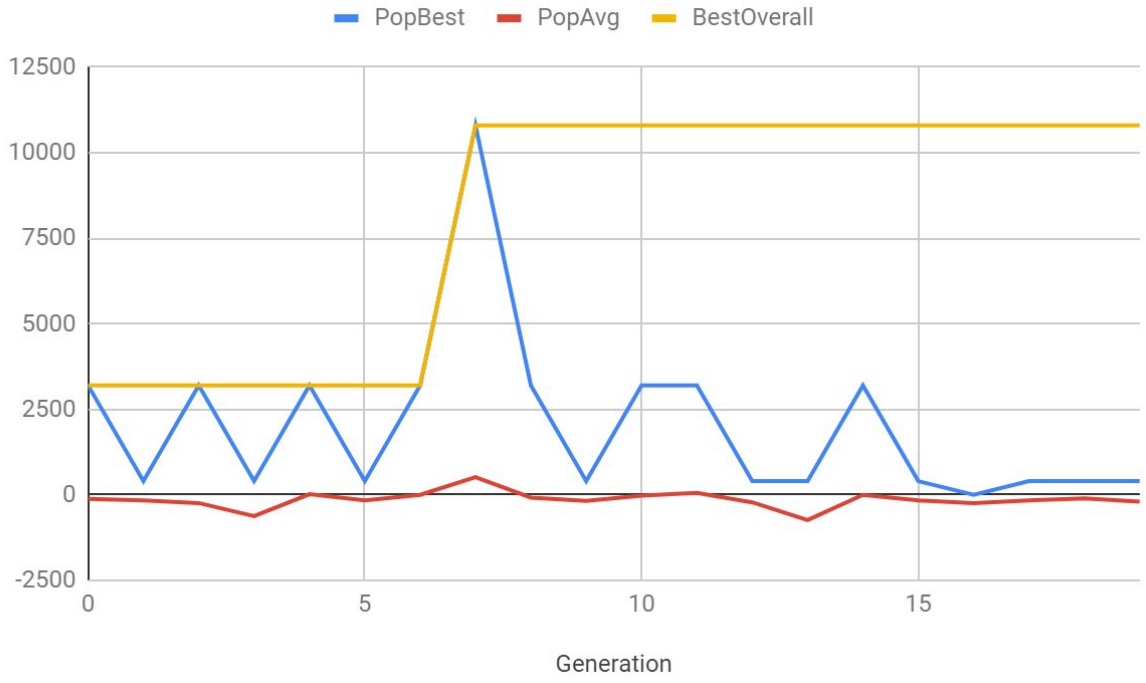
Caso de teste #5:

Population Size: 20
Num Generations: 20
Mutation Probability: 0.4
Crossover Probability: 0.01
Tournament Size: 5
Elitist: No



Caso de teste #6:

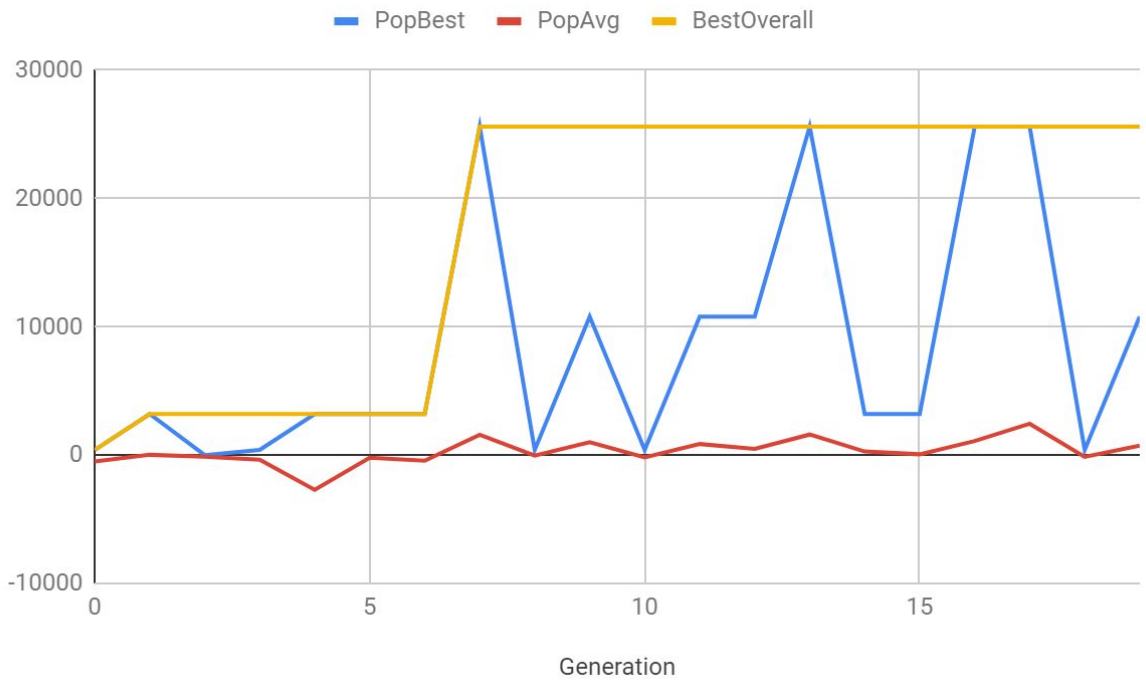
Population Size: 20
Num Generations: 20
Mutation Probability: 0.8
Crossover Probability: 0.01
Tournament Size: 5
Elitist: No



Nota: Para este caso de teste verificámos que, tal como esperado, quanto menor for a probabilidade de mutação, melhores são os resultados obtidos na função de fitness visto que os melhores cromossomas não vão ser alterados de geração em geração.

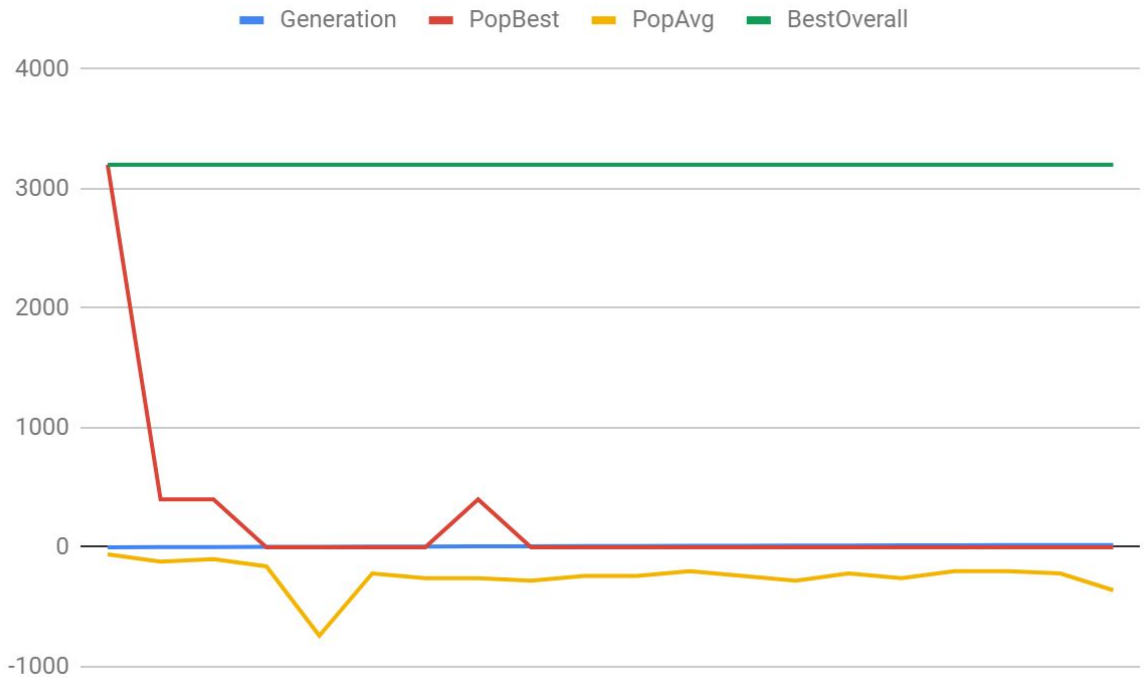
Caso de teste #7:

Population Size: 20
Num Generations: 20
Mutation Probability: 0.01
Crossover Probability: 0.05
Tournament Size: 5
Elitist: No



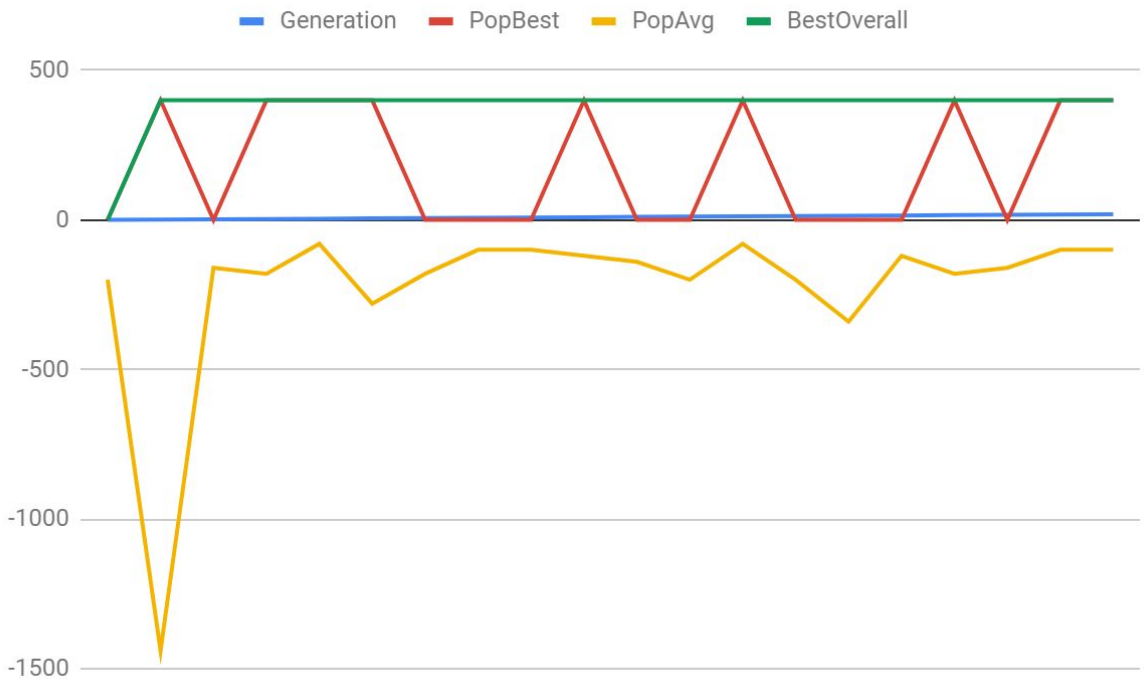
Caso de teste #8:

Population Size: 20
Num Generations: 20
Mutation Probability: 0.01
Crossover Probability: 0.4
Tournament Size: 5
Elitist: No



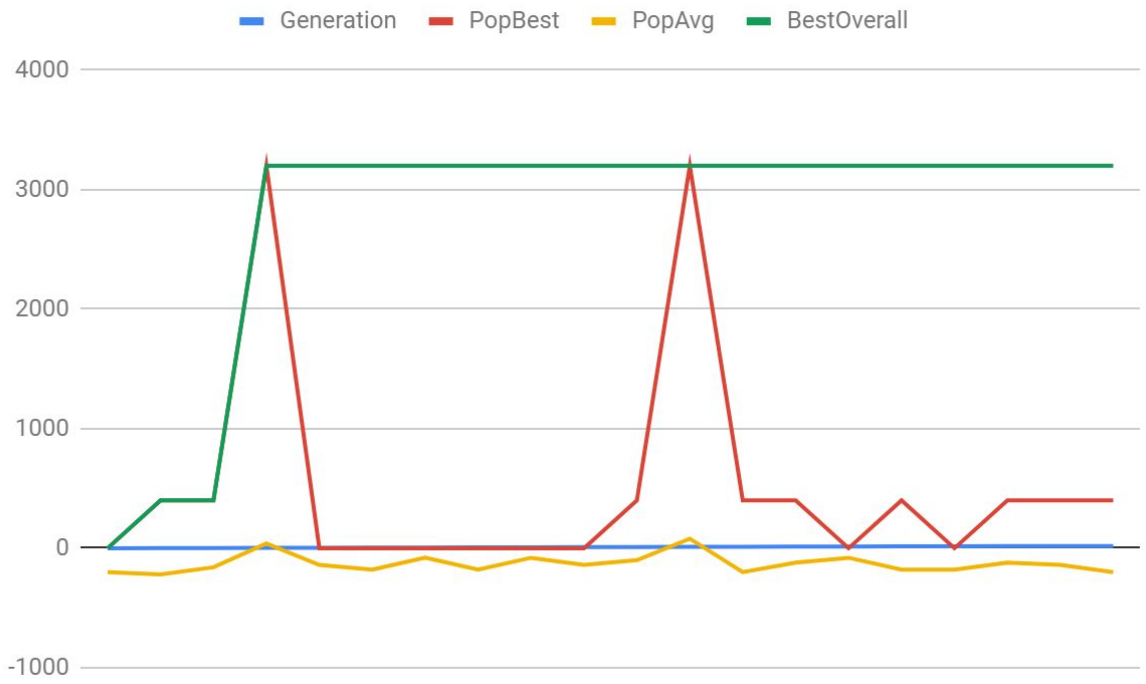
Caso de teste #9:

Population Size: 20
Num Generations: 20
Mutation Probability: 0.01
Crossover Probability: 0.8
Tournament Size: 5
Elitist: No



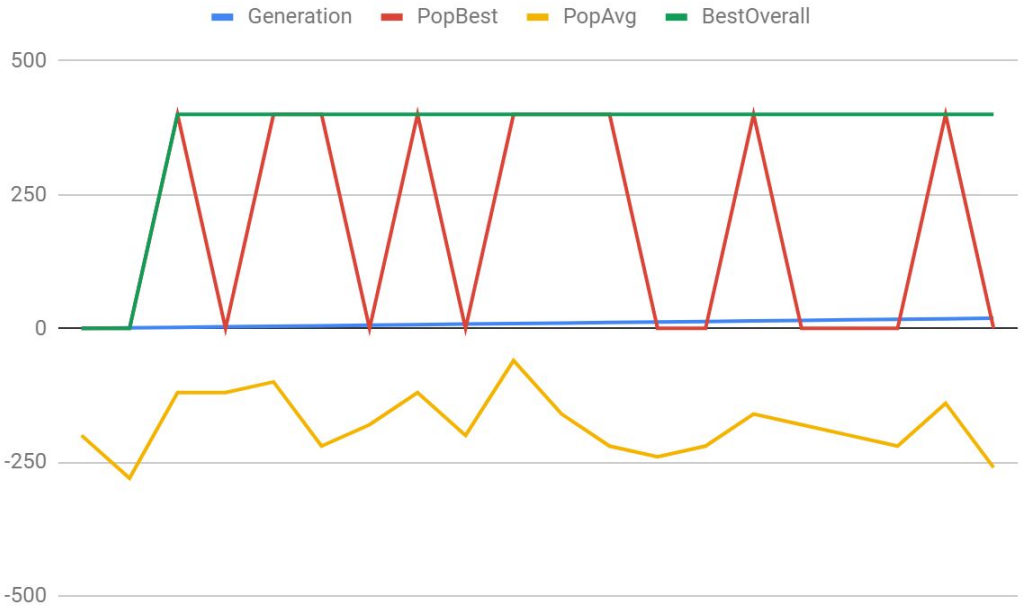
Caso de teste #10:

Population Size: 20
Num Generations: 20
Mutation Probability: 0.01
Crossover Probability: 0.01
Tournament Size: 5
Elitismo: Yes
Size: 1



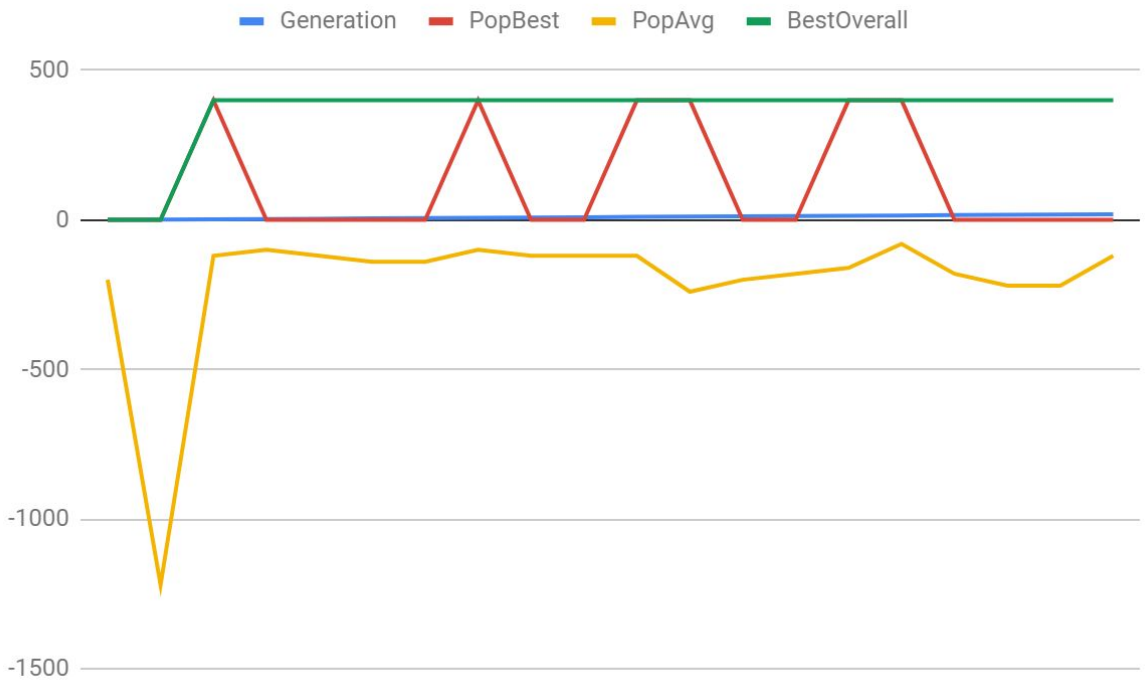
Caso de teste #11:

Population Size: 10
Num Generations: 20
Mutation Probability: 0.01
Crossover Probability: 0.01
Tournament Size: 5
Elitismo: Yes
Size: 3



Caso de teste #12:

Population Size: 10
Num Generations: 20
Mutation Probability: 0.01
Crossover Probability: 0.01
Tournament Size: 5
Elitismo: Yes
Size: 5



Problemas e soluções encontradas

- Problema em usar os dados obtidos da experimentação do algoritmo visto que para algumas versões do Excel os dados ficavam todos concatenados na mesma coluna o que dificultava a análise e produção dos gráficos correspondentes. Após pesquisar, verificámos que este problema seria resolvido com a alteração do código do professor separando os valores de cada parâmetro com um ponto e vírgula.
- Para certos valores de população (por exemplo 50), o programa indicava um erro de índice no array de uma classe do resultado do jogo. Após a análise do código, não conseguimos resolver este problema.

Conclusão

Tendo em conta uma análise pormenorizada aos resultados obtidos experimentalmente, podemos tirar algumas conclusões.
Os algoritmos genéticos são favorecidos quando a probabilidade de mutação é baixa, uma vez que é mais improvável alterar os genes resultantes do método de seleção utilizado. Um valor alto deste operador, poderá fazer com o que o algoritmo funcione de uma forma um bocado “aleatória”.
Em relação ao crossover, uma probabilidade alta deste operador irá levar a uma introdução mais rápida de novos indivíduos na população. No entanto, se esta for muita alta, poderá haver indivíduos com uma boa aptidão a serem substituídos.
Por fim, relativamente ao método de substituição utilizado (“Elitismo”), observamos que deve ser utilizado, uma vez que preserva os n indivíduos com melhor aptidão de cada população, para a geração seguinte.
Concluindo, devemos procurar utilizar taxas baixas de mutação e taxas moderadas de crossover. Posteriormente, devemos optar por utilizar um método de substituição de elitismo.

Bibliografia

- Material de apoio da cadeira
- Artificial Intelligence: A Modern Approach
- Inteligência Artificial: Fundamentos e Aplicações
- Wikipédia