Instituto Superior de Engenharia de Lisboa

ISEL INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Inteligência Artificial e Sistemas Cognitivos

Inteligência Artificial e Sistemas Cognitivos

Mestrado em Engenharia Informática e Multimédia

Pedro Gonçalves, 45890

Semestre de Inverno, 2021/2022

1. Introdução

Este projeto almejará explorar técnicas de aprendizagem automática para resolver determinados problemas. As técnicas estudadas e implementadas neste projeto residem no âmbito das redes neurais, aprendizagem por reforço, algoritmos genéticos e raciocínio automático para otimização e para planeamento.

Conforme as diferentes técnicas de aprendizagem automática vão sendo abordadas, determinados problemas vão sendo introduzidos e implementados.

2. Inteligência Artificial e Sistemas Cognitivos

Durante muito tempo, assumiu-se que a inteligência artificial (***AI***) não passava de um mito, um produto da ficção científica, que nunca se tornaria realidade. As máquinas jamais poderiam apresentar características que se assemelhem à inteligência, uma qualidade exclusivamente dos humanos. Contudo, as primeiras revoluções industriais deram origem a equipamentos que substituiram a mão de obra em várias áreas, realizando, com maior eficácia e menor custo, o trabalho de muitos homens. Hoje, é sabido que o potencial das máquinas é ainda mair.

Atualmente, o conceito de inteligência artificial consiste na aprendizagem das máquinas com base na experiência, e ajustem a novas entradas de dados de modo a realizarem determinadas tarefas como seres humanos.

Mais concretamente, o sistemas cognitivos são sistema capazes de utilizar a informação do ambiente envolvente, de forma autónoma, para tomar decisões. Um sistema diz-se racional, se tomar a decisão correta, dado o conhecimento que possui, ou seja, apresenta a capacidade de agir, no sentido de conseguir o melhor resultado possível perante os objectivos que se pretende atingir.

Existem imensas técnicas para proporcionar essa capacidade a uma máquina, das quais algumas serão abordadas neste projeto da seguinte forma organizada:

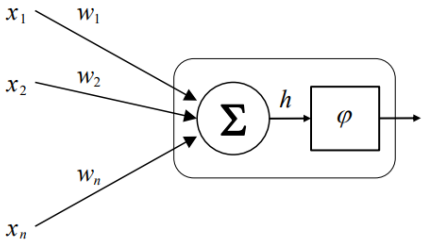
- Introdução teórica do tema, técnica ou algoritmo;
- Explicação do(s) problema(s) a resolver;
- Implementação;
- Testes e resultados.

3. Redes Neurais

3.1. Introdução Teórica

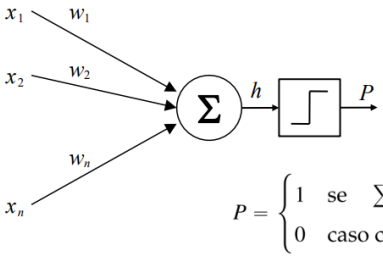
Começar-se-á por estudar as redes neurais. As redes neurais baseiam-se no modelo biológico composto por neurónios interligados que comunicam através de sinapses.

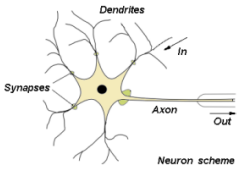
As sinapses controlam a influência que a saída de um neurónio tem sobre o seguinte neurónio, através da permeabilidade à passagem do impulso eléctrico. A variabilidade da permeabilidade de cada sinapse é determinante para a aprendizagem.



Um neurónio artificial pode ser representado por um conjunto de entradas $\mathbf{x_i}$, em que cada entrada possui um peso associado $\mathbf{w_i}$, que simboliza a permeabilidade da sinapse, sendo a soma ponderada das entradas pelos pesos aplicada a uma função de activação **φ**, a qual determina se é produzida resposta ou não.

O modelo do perceptrão (Rosenblatt, 1957) admite valores de entrada binários (podem ser resultantes de uma combinação de entradas através de uma função lógica) e um valor de saída binário.

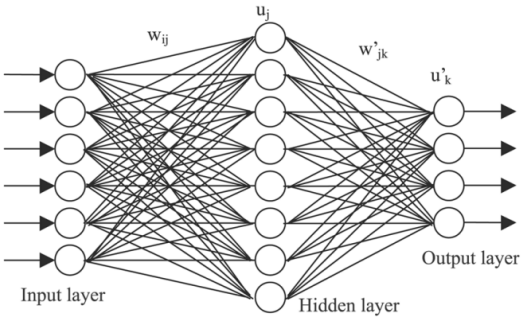


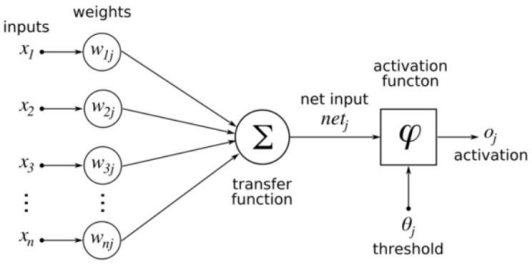


$$P = \begin{cases} 1 & \text{se} \quad \sum_i w_i \times x_i > T \\ 0 & \text{caso contrário} \end{cases}$$

O modelo do perceptrão permite resolver problemas linearmente separáveis. Um problema é linearmente separável se existirem pesos $\mathbf{w_i}$ e pendor **b** que definam uma fronteira linear (geometricamente) entre a região da resposta excitadora e a região da resposta inibitória. Exemplos desse tipo de problemas seriam as operações **AND** ou **OR**.

Para resolver problemas mais complexos, será necessário oragnizar neurónios em modelos multi-camada. As redes neurais multi-camada apresentam sempre uma camada de input e uma de output. Entre elas, poderão existir um número de camadas escondidas arbitrário, consoante a dimensão do problema que se pretende resolver.



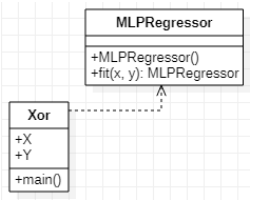


3.2. Problema XOR

Ao contrário das operações **AND** e **OR** (como foi abordado), a operação **XOR** implica a utilização de pelo menos uma camada escondida. Para este problema pretende implementar uma rede neural que aceite como dados de entrada dois bits, e produza o resultado **XOR** (***bitwise***) num único bit. Isto implica que a camada de input do modelo neural tenha dois neurónios, e a camada de output tenha um neurónio.

3.2.1. Implementação

Para implementar este problema, e qualquer outro problema deste projeto que envolva redes neurais, utilizar-se-á a biblioteca **SciKit-Learn** para a linguagem **Python**. Observe-se a figura que ilustra o programa implementado. **X** é um array com o conjunto de treino que consiste em **[[-1, -1], [-1, 1], [1, -1], [1, 1]]** e **Y** é um array com o output respetivo **[-1, 1, 1, -1]** (target data).



A rede neural contará com duas camadas escondidas, uma com **4** neurónios e outra com **2** neurónios. Mas porque não apenas uma camada com um neurónio? E porque não centenas de camadas com centenas de neurónios?

Ora, seria incorreto pensar que quantas mais camadas e neurónios, melhor, dado que existe o fenómeno de ***overfitting***. Este fenómeno ocorre quando a rede neural tenta aprender demasiados detalhes, incluindo o ruído, causando um desempenho muito pobre. No caso de utilizar apenas uma camada com um neurónio, correr-se-ia o risco de a rede neural não aprender detalhes suficientes, resultanto também num desempenho pobre. Este fenómeno designa-se ***underfitting***.

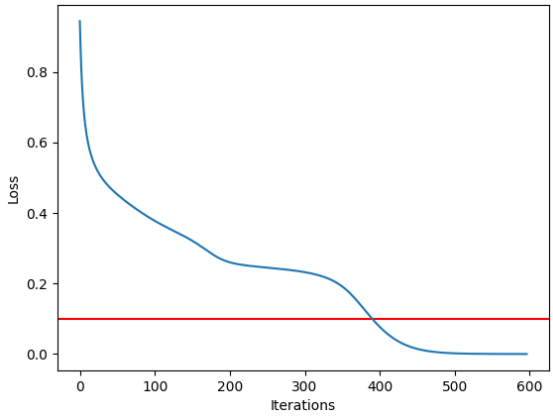
Utilizar-se-á um optimizador ***SGD*** (***Stochastic Gradient Descent***) que atualiza os pesos com recurso ao algoritmo de ***back-propagation***.

3.2.2. Testes e Resultados

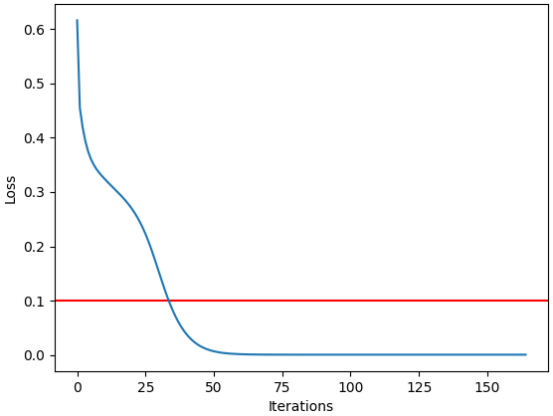
A classe **MLPRegressor** aceita parâmetros como o momento, a taxa de aprendizagem, a apresentação aleatória, a função de activação e o algoritmo e resolução. O primeiro passo é estabelecer parâmetros padrão, para que se possa testá-los de forma individual e sucessiva. Os parâmetros padrão são:

- Taxa de aprendizagem = 0.05;
- Momento = 0;
- Aleatoriedade = False;
- Função de activação = "tanh" (visto que os dados estão em codificação bipolar **[-1 ,1]**);
- Algoritmo de resolução = "sgd";

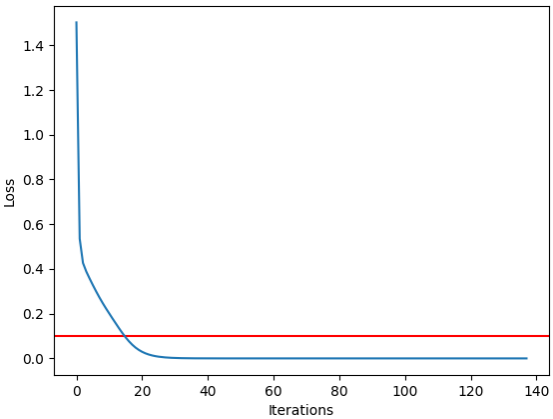
Dados estes parâmetros, e variando a taxa de aprendizagem obtêm-se resultados interessantes. Observando os gráficos da página seguinte, que representam a curva do erro associada à aprendizagem, conclui-se que a taxa de erro de **0.1** é atingida mais rapidamente, à medida que a taxa de aprendizagem aumenta. Isto acontece porque o modelo não fica preso em mínimos locais (pelo menos com tanta frequência) e consegue "saltar" e ultrapassá-los, pois pratica uma aprendizagem mais intensa.



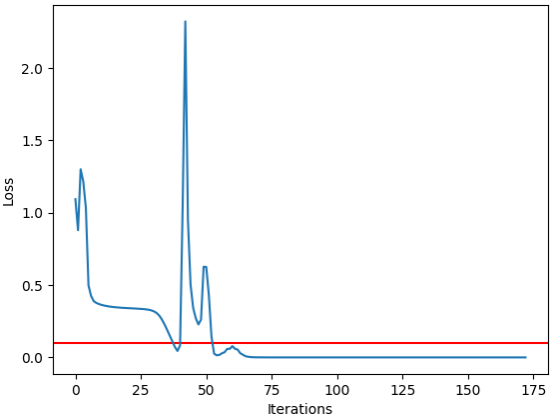
3 - Taxa de Aprendizagem = 0.05



2 - Taxa de Aprendizagem = 0.25



4 - Taxa de Aprendizagem = 0.5



1 - Taxa de Aprendizagem = 1

Contudo, ao aumentar a taxa de aprendizagem em demasia, esses saltos tornam-se imprevisíveis, no sentido em que a aprendizagem torna-se mais desorganizada e menos metódica. Na figura **4**, em que a taxa de aprendizagem é **1**, essa desorganização e imprevisibilidade faz-se notar. Por volta da iteração **32**, é descoberto um mínimo local, do qual o modelo sai imediatamente de forma desamparada (a taxa de erro sobe imenso porque a saída de um mínimo local implica o aumento da mesma), acabando por convergir num mínimo menor, na iteração **52**.

No caso de um taxa de aprendizagem igual a **2**, a representação gráfica é desnecessária pois o modelo torna-se inutilizável. Os "saltos" passam a ser tão grandes, e a aprendizagem tão desorganizada e imprevisível, que a descoberta de mínimos com uma taxa de erro abaixo de **0.1** é muito pouco provável.

Em suma, conclui-se que o aumento da taxa de aprendizagem diminui o número de iterações necessárias para atingir a mesma taxa de erro, resultando num tempo de aprendizagem mais reduzido. Contudo, ao aumentar esse parâmetro em demasia, a aprendizagem e a procura de mínimos torna-se incerta e instável, resultando numa taxa de erro de convergência mais elevada.

A tabela da página seguinte enumera algumas execuções para as diferentes taxas de aprendizagem (r), indicando o número de iterações até à convergência. No caso de a convergência ocorrer acima da taxa de erro de **0.1**, a execução é marcada como **ND** (não definido).

Execução	r = 0.05	r = 0.25	r = 0.5	r = 1	r = 2
1	163	23	ND	9	ND
2	76	13	33	31	ND
3	89	28	48	21	ND
4	ND	12	33	22	ND
5	158	246	41	17	ND
6	274	34	65	57	ND
7	133	ND	20	37	ND
8	212	29	58	ND	ND
9	201	19	41	25	ND
10	391	35	16	39	ND
Média:	189	49	39	29	ND

Ora, se ao aumentar a taxa de aprendizagem em demasia prejudica seriamente o desempenho do modelo, como é que se pdoe tirar o maior partido de baixas taxas de aprendizagem? Para responder a essa questão, introduzir-se-á o parâmetro **momento**. Para um momento igual a **0.5**, obtêm-se valores bastante inferiores aos obtidos com um momento igual a **0**. Ou seja, com momento, o modelo necessita de menos iterações para convergir para um valor abaixo dos **0.1**. Observe-se a tabela.

Execução	r = 0.05	r = 0.25	r = 0.5	r = 1	r = 2
1	153	42	24	ND	ND
2	70	51	14	ND	ND
3	65	14	16	ND	ND
4	197	39	21	ND	ND
5	89	ND	17	ND	ND
6	ND	ND	13	ND	ND
7	50	65	15	ND	ND
8	55	34	54	ND	ND
9	65	33	9	ND	ND
10	64	ND	60	ND	ND
Média:	90	40	24	ND	ND

O momento oferece ao modelo uma característica que lhe permite oscilar, da mesma forma que uma bola de basquetebol salta no chão (ealizando saltos cada vez mais pequenos até parar totalmente no chão). A tabela demonstra essa característica melhora o modelo quando se usam taxas de aprendizagem reduzidas. Contudo, quando se usam taxas de aprendizagem mais elevadas, a desorganização e instabilidade apenas é ampliada, e os maus resultados que, previamente, se verificavam apenas para uma taxa de aprendizagem de **2**, tamém se verificam agora para uma taxa de aprendizagem de **1**.

Na tabela seguinte, verifica-se que, ao utilizar um momento igual a **1**, os resultados tornam-se instáveis para praticamente todas as taxas de aprendizagem. Conclui-se então que o melhor valor de momento é realmente o **0.5**.

Execução	r = 0.05	r = 0.25	r = 0.5	r = 1	r = 2
1	ND	ND	ND	ND	ND
2	63	ND	ND	ND	ND
3	28	ND	ND	ND	ND
4	ND	17	ND	ND	ND
5	22	16	ND	ND	ND
6	47	23	ND	ND	ND
7	ND	ND	ND	ND	ND
8	ND	9	14	ND	ND
9	ND	ND	12	ND	ND
10	11	19	ND	ND	ND
Média:	34	17	13	ND	ND

O próximo parâmetro a testar é o de aleatoriedade (***shuffle***). Como se observa na tabela seguinte, este parâmetro não parece influenciar os resultados de forma minimamente impactante.

Execução	r = 0.05	r = 0.25	r = 0.5	r = 1	r = 2
1	36	27	29	ND	ND
2	110	187	12	ND	ND
3	183	10	52	ND	ND
4	93	10	136	ND	ND
5	117	17	15	ND	ND
6	84	14	11	ND	ND
7	35	24	7	ND	ND
8	42	8	16	ND	ND
9	ND	44	9	ND	ND
10	226	30	14	ND	ND
Média:	103	37	30	ND	ND

Contudo é bom utilizar este parâmetro para misturar as amostras em cada iteração, para que o modelo não se habitue inadvertidamente a determinados padrões.

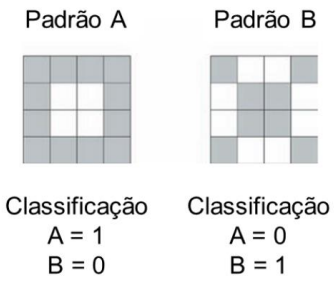
Por fim, testar-se-á outro tipo de codificação dos dados. Ao invés de utilizar uma codificação bipolar **[-1, 1]**, utilizar-se-á uma codificação binária **[0, 1]**. Logo, ter-se-á também de alterar a função de ativação, visto que a tangente hiperbólica resulta em valores entre **-1** e **1**. Ao testar esta codificação com a tangente hiperbólica, perceber-se-ia imediatamente que a taxa de erro dispara. Por isso, utiliziar-se-á a função **relu**.

Execução	r = 0.05	r = 0.25	r = 0.5	r = 1	r = 2
1	28	43	ND	ND	ND
2	53	68	ND	ND	ND
3	ND	69	ND	ND	ND
4	41	98	ND	ND	ND
5	31	3	ND	35	ND
6	47	ND	9	ND	ND
7	ND	ND	ND	ND	ND
8	20	10	12	ND	ND
9	118	2	ND	ND	ND
10	83	ND	28	ND	ND
Média:	53	41	16	35	ND

Verifica-se que no caso em que a taxa de aprendizagem é **0.05**, para a qual o número de iterações até à convergência costuma ser elevado, reduziu bastante. Em geral os resultados são melhores, mas por outro lado a convergência deu-se mais vezes acima da taxa de erro **0.1**.

3.3. Problema dos Padrões

Este problema consiste em ensinar a rede neural a reconhecer os padrões da figura e, se não se tratar de nenhum deles, deverá reconhecê-lo também. Ou seja, a rede neural receberá como dados de entrada um array bidimensional com quatro posições em ambas as direções (um quadrado, portanto), como mostra a figura. Na camada de saída haverão dois neurónios, um para cada padrão. Se a rede neural não detetar nenhum dos padrões deverá manter os neurónios inativos.

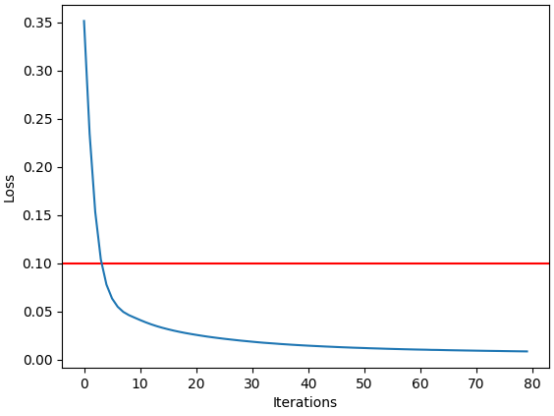


3.3.1. Implementação

Como neste exercício, o conjunto de domínio é bastante mais vasto (existem múltiplas combinações para formar novos padrões), será necessário, primeiro que tudo, gerar uma função capaz de gerar padrões aleatórios. O conjunto de treino **X** será composto por, aproximadamente, por três quartos de padrões aleatórios, e um quarto de padrões predefinidos (**A** e **B**, em iguais quantidades), Quanto ao conjunto de dados de saída (***target data***), será composto por **[1, 0]** e **[0, 1]**, no caso dos padrões **A** e **B**, respetivamente, e por **[0, 0]**, no caso de um padrão diferente.

3.3.2. Testes e Resultados

Com a rede neural treinada, e com recurso a um conjunto de teste composto por um padrão A, um padrão B e um padrão aleatório, verifica-se a veracidade das predições da mesma.



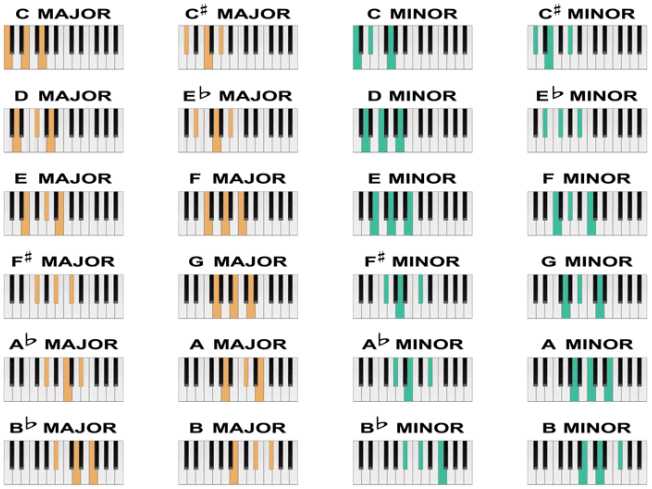
Os resultados obtidos são corretos, e a aprendizagem converge para uma taxa de erro aceitável (à volta de **0.02**). Verifica-se também que é atingida uma taxa de erro de **0.1** nas primeiras **10** iterações.

Depois de alguns testes, conclui-se que a utilização duas camadas escondidas, uma com **16** neurónios e outra com **8** neurónios é a abordagem mais eficaz (para evitar ***overfitting*** ou ***underfitting***).

3.4. Problema dos Acordes Musicais

Este problema procura reconhecer acordes tocados no piano (a informação das notas tocadas passa para o computador por **MIDI** e é interpretada pela biblioteca **Pygame** em **Python**), ou no teclado do computador (as teclas pressionadas são também reconhecidas pela mesma biblioteca).

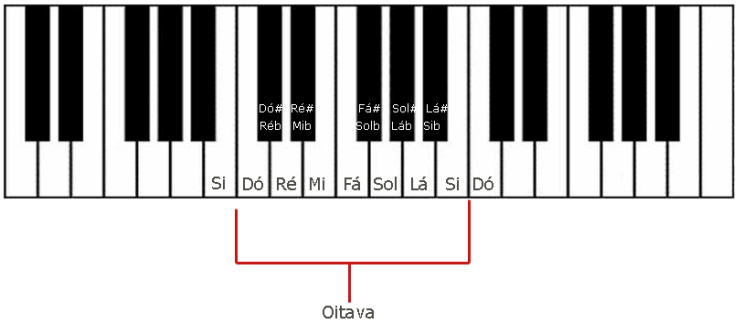
Na música (teoria musical), os acordes fundamentais são compostos por 3 notas. A tónica, a terceira e a quinta. Naturalmente, existem 12 notas diferentes (**Dó**, **Dó Sustenido**, **Ré**, **Ré Sustenido**, **Mi**, **Fá**, **Fá Sustenido**, **Sol**, **Sol Sustenido**, **Lá**, **Lá Sustenido** e **Si**), pelo que seria de esperar que existiriam **12** acordes diferentes (sendo cada um destes acordes a tónica de cada acorde). E é verdade. Contudo, existem acordes maiores e menores. Os acordes menores são semelhantes aos menores, com a excepção de uma das notas. Tome-se como exemplo o acorde de dó maior (**C MAJOR**) e o acorde de dó menor (**C MINOR**). A tónica e a quinta mantêm-se. Apenas a terceira muda, sendo que no acorde menor desce meio tom (uma tecla).



	Inglês	Português	
C	C# or Db	Dó	Dó Sustenido
D	D# or Eb	Ré	Ré Sustenido
E	F	Mi	Fá
F# or Gb	G	Fá Sustenido	Sol
G# or Ab	A	Sol Sustenido	Lá
A# or Bb	B	Lá Sustenido	Si

Note-se a equivalência da terminologia portuguesa e inglesa representada na tabela , referente à denominação dos acordes. Com toda esta informação, é possível concluir o seguinte:

- Em qualquer acorde, a quinta dista **7** meio-tons da tónica;
- Nos acordes maiores, a terceira dista **4** meio-tons da tónica;
- Nos acordes menores, a terceira dista **3** meio-tons da tónica;
- Existem **24** acordes diferentes (maiores e menores)Acrescenta-se também que uma **oitava** é a distância entre duas notas iguais, isto é, de **Dó** a **Dó**, por exemplo. Por isso, uma oitava conta com **12** notas diferentes. Isto significa que o espaço de uma oitava é suficiente para representar todos os **24** acordes.

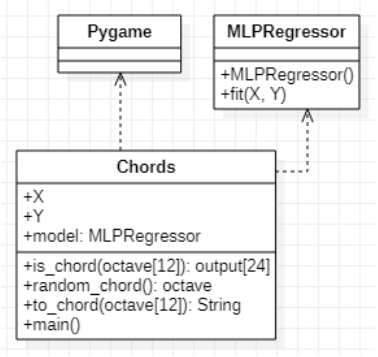


3.4.1. Implementação

Uma oitava será representada pela seguinte lista: **[*, *, *, *, *, *, *, *, *, *, *, *]** com dimensão **12**. Os dados de entrada terão este formato, em que as posições ativadas (com valor **1**) representarão notas a soar, e as posições desativadas (com valor **0**) representarão notas em silêncio. Dado que haverá **24** acordes possíveis, a camada de saída contará com **24** neurónios, um para cada acorde. Os primeiros **12** serão maiores (de **Dó** a **Si**) e os segundos **12** serão menores (também de **Dó** a **Si**).

O objetivo da rede neural a implementar é, com base na oitava de entrada, concluir acerca do acorde que está a ser tocado. Por exemplo, o acorde Dó Maior (que contém a tónica Dó, a terceira Mi, e a quinta Sol), se fosse tocado, a entrada da rede neural seria **[1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0]** e a predição correta seria **[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]**.

Como já foi referido, para recolher informação acerca das teclas tocadas tanto no piano, como no teclado do computador, utilizar-se-á a biblioteca **Pygame** do **Python**.



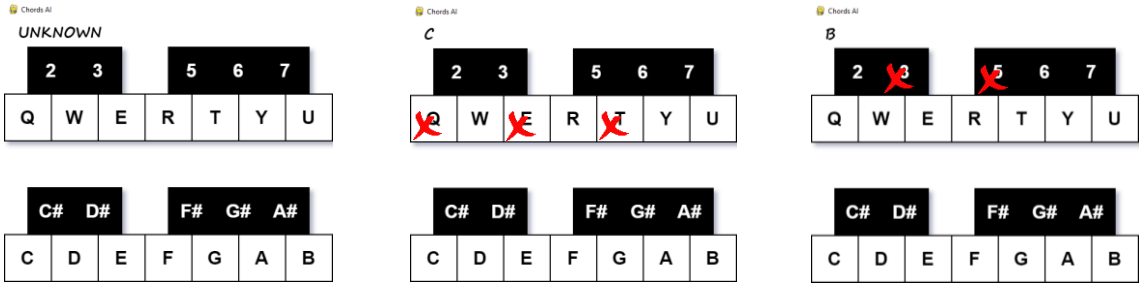
Será necessária uma função que, com base numa oitava, seja capaz de gerar um ***array*** com **24** posições indicando o acorde que está a ser tocado, para mais facilmente gerar o conjunto de dados **Y** (***target data***). Será também necessária uma função para gerar acordes aleatórios e uma função para, com base numa oitava, identificar o acorde que está a ser tocado (o nome do acorde).

Será então desenvolvido um pequeno programa em **Python** para treinar o modelo e para ouvir eventos no piano (ou no teclado do computador) relativos a teclas premidas, mostrando sempre o acorde que está a ser tocado.

Para o modelo da rede neural serão escolhidas duas camadas escondidas, uma com **128** neurónios e outra com **64** neurónios, será escolhida a função de ativação **relu** (codificação binária), uma taxa de aprendizagem de **0.05** e um momentum de **0.5**.

3.4.2. Testes e Resultados

Ao executar o programa, o modelo é treinado e surge uma pequena interface gráfica que mostra a localização de cada nota no teclado do computador, bem como o acorde que está a ser tocado no momento. Na figura da esquerda, é mostrado **"UNKNOWN"**, visto que nenhuma tecla está a ser premida e nenhum acorde está a soar. Contudo, na figura do meio observa-se que, ao premir a tecla **Q**, a tecla **E** e a tecla **T**, a rede neural identifica o acorde **C MAJOR**.



Um facto curioso é que na figura da direita, as únicas notas que estão a ser premidas são **Ré Sustenido** e **Fá Sustenido**, ou seja, a tecla **3** e a tecla **5**. Ora, um acorde precisa de pelo menos três notas para que se justifique, mas o modelo assumiu que não é esse o caso, e tomou a decisão (errada, mas não deixa de ser razoável e interessante) de identificar o acorde Si, apesar de faltar a tónica (o próprio **Si**). Esta decisão é errada porque, com base nestas duas notas, haveria outro acorde possível. O acorde Ré Sustenido Menor conta com as seguintes notas: **Ré Sustenido** como tónica, **Fá Sustenido** como terceira e **Lá Sustenido** como quinta.

4. Algoritmos Genéticos e de Otimização

Em que consiste a otimização? A otimização consiste na seleção da melhor opção a partir de um conjunto de opções possíveis de acordo com um critério de avaliação dessas opções. Dependendo do algoritmo, essa decisão poderá almejar maximização ou minimização de um determinado parâmetro, de acordo com o critério de avaliação. Dependendo também do algoritmo, o critério de avaliação poderá ser um dos seguintes:

- Medida de ganho (***performance***);
- Medida de perda (***loss***);
- Medida de adequação (***fitness***).

Existem vários métodos computacionais de otimização meta-heurísticos, entre os quais:

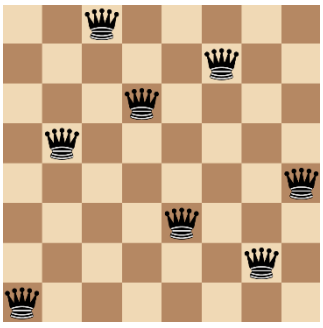
- Procura local sôfrega (**Hill-climbing**);
- Têmpera simulada (**Simulated Anealing**);
- Algoritmos genéticos.

A resolução de problemas de otimização consiste em descobrir qual a melhor configuração de parâmetros de modo a maximizar uma função de valor ou de adequação. Isto significa que o resultado final será uma configuração de parâmetros (um estado).

- **Estado -** Representa uma configuração;
- **Transição -** Representa uma transformação de estado;
- **Valor -** Função de valor de estado;
- **Solução -** Estado final.

4.1. Introdução ao Problema NQueens

Este problema constitui um tabuleiro **n** x **n** pelo qual se deslocarão **n** rainhas. Inicialmente, as rainhas estarão aleatoriamente distribuídas pelo tabuleiro, garantindo-se apenas que há uma e apenas uma em cada linha. Para resolver o problema, as rainhas deverão deslocar-se pela tabuleiro (neste caso, deslocam-se apenas pela sua linha) de forma a atingir zero colisões. Uma rainha está em colisão com outra se for possível traçar uma reta na horizontal, vertical ou diagonal (**45º**) entre elas. A figura ilustra um tabuleiro com **8** rainhas (e por isso, com uma dimensão de **8** x **8**), com zero colisões.



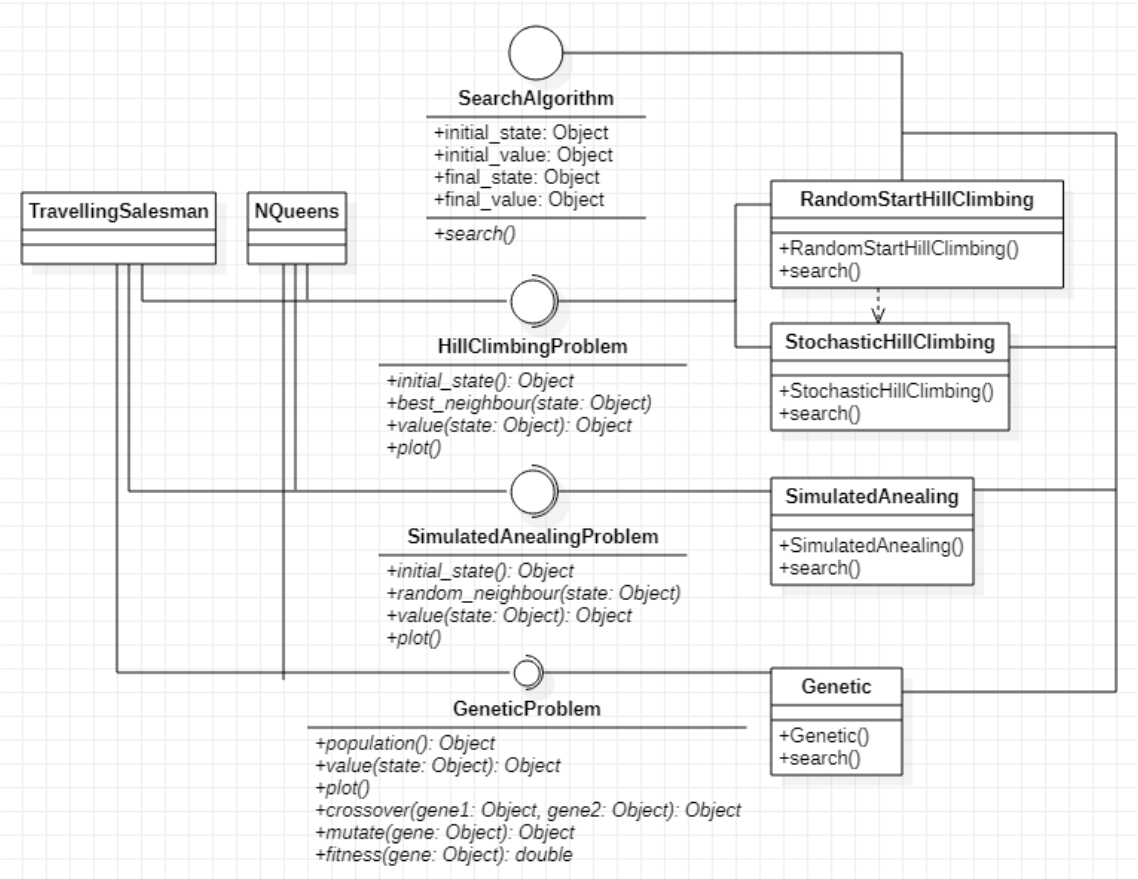
Na implementação deste problema, será necessário um método para detectar o número de colisões no tabuleiro e um para gerar um tabuleiro aleatório. O problema será representado por uma lista com dimensão **n**, em que cada posição representa a coluna de cada rainha. Por exemplo, no caso da disposição da figura, o conjunto seria: **[2, 5, 3, 1, 7, 6, 4, 0]**.

4.2. Introdução ao Problema Travelling-Salesman

Este problema é representado por um espaço bidimensional, de comprimento igual à largura (tal como no problema **NQueens**), no qual se apontarão **n** cidades. Cada cidade conta com uma coordenada **x** e uma coordenada **y**. Por outras palavras, este problemas constitui um conjunto de cidades (tuplos de coordenadas) em que a ordem é relevante, pois será a ordem pela qual o **Travelling Salesman** visitará as cidades. Contudo, ele pretende percorrer a menor distância possível, pelo que, na resolução do problema, ter-se-á de alterar a ordem das cidades, mas nunca os valores das coordenadas das cidades.

4.3. Implementação

Para implementar o problema do **NQueens** e o problema do **Travelling-Salesman**, e forma a que seja possível aplicar várias técnicas de otimização diferentes, é necessária uma abordagem estruturada e metódica. Observe-se o diagrama de classes da figura seguinte.



Ambos os problemas implementam as interfaces **HillClimbingProblem**, **SimulatedAnelingProblem** e **GeneticProblem**. Esta abordagem garante que tanto o **NQueens** como o **Travelling-Salesman**, apresentarão os métodos necessários para que sejam aplicadas as três técnicas de otimização.

As classes que implementam **SearchAlgorithm** deverão ser capazes de aceitar um problema de um determinado tipo, e após a pesquisa, armazenar o estado final em memória, para que seja analisado.

4.4. HillClimbing

A primeira técnica de otimização que será explorada é o **HillClimbing**. As duas técnicas que serão estudadas e implementadas serão o **Stochastic HillClimbing** e o **Random Start HillClimbing**.

4.4.1. Introdução Teórica

O **HillClimbing** trata-se de um algoritmo de procura local sôfrega. O **Stochastic HillClimbing**, em específico, caracteriza-se:

- Pela escolha aleatória entre sucessores que aumentam o valor de estado;
- Pela convergência mais lenta que HillClimbing simples;
- Por poder encontrar melhores soluções (consoante a tipologia de espaço de estados);

A grande diferença relativamente ao **HillClimbing** simples é que se existirem vários movimentos ***uphill*** possíveis, ele não escolherá sempre o mesmo, visto que essa escolha é efetuada de modo aleatório. Este processo passa por averiguar todas os movimentos possíveis num determinada espaço de estados e armazenar todos os melhores. Do grupo dos melhores deverá sortear apenas um único movimento.

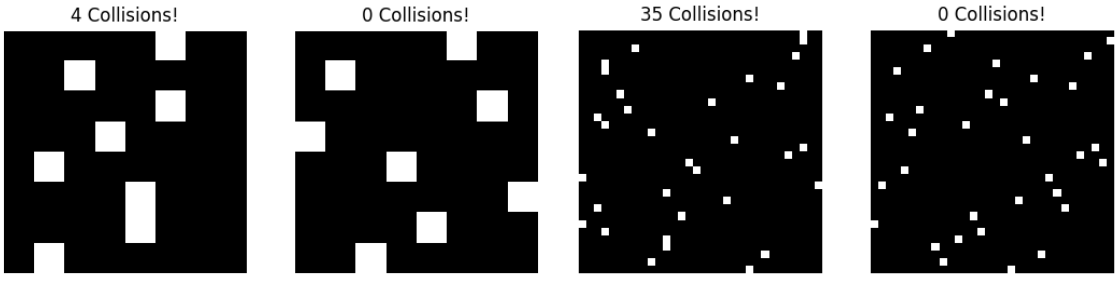
O **Random Start HillClimbing** caracteriza-se por proceder à pesquisa efetuada pelo **Stochastic HillClimbing** múltiplas vezes. Contudo, deverá ser destacada que em cada iteração, o espaço de estados é baralhado, isto é, o ponto de partida é diferente mas a dimensão do problema é a mesma.

No caso do problema **NQueens**, no início de cada iteração, cada rainha é colocada numa nova coluna aleatória, mantendo sempre o número de rainhas inicial.

No caso do Problema **Travelling Salesman**, no início de cada iteração, a ordem das cidades é alterada, mas as cidades mantêm-se as mesmas. Desde já, afirma-se que o espectável é que o algoritmo **Random Start HillClimbing** seja mais eficaz, visto que executará o **Stochastic HillClimbing** múltiplas vezes e apresentará como resultado, o resultado da melhor iteração.

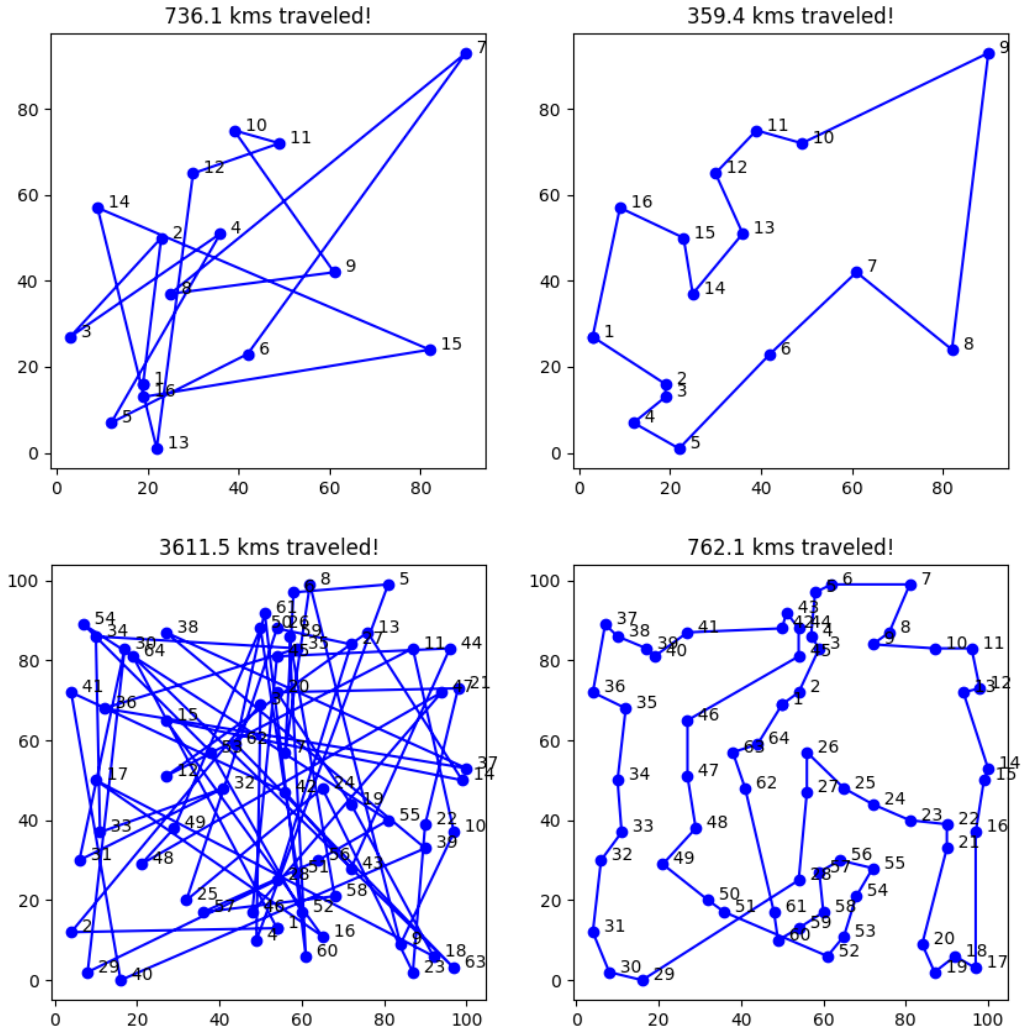
4.4.2. Resultados e Testes

Ao aplicar o **Stochastic HillClimbing** no problema das **NQueens**, obtêm-se os seguintes resultados:



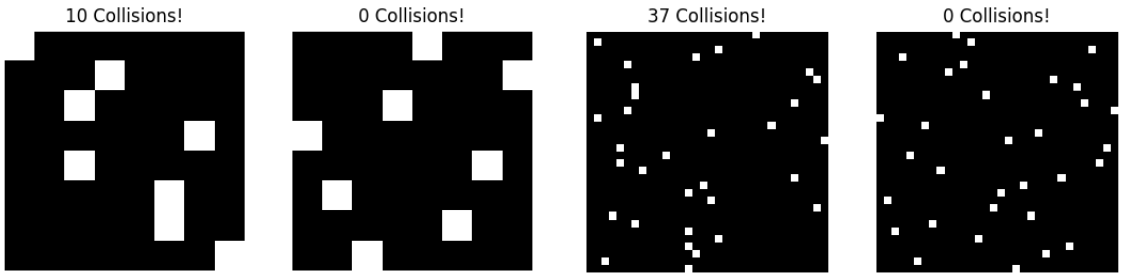
Verificam-se os resultados ideais, apesar de, para dimensões maiores (como as da figura da direita, com **32** rainhas), seja mais demorado.

Por sua vez, o problema do **Travelling Salesman** apresenta os seguintes resultados.



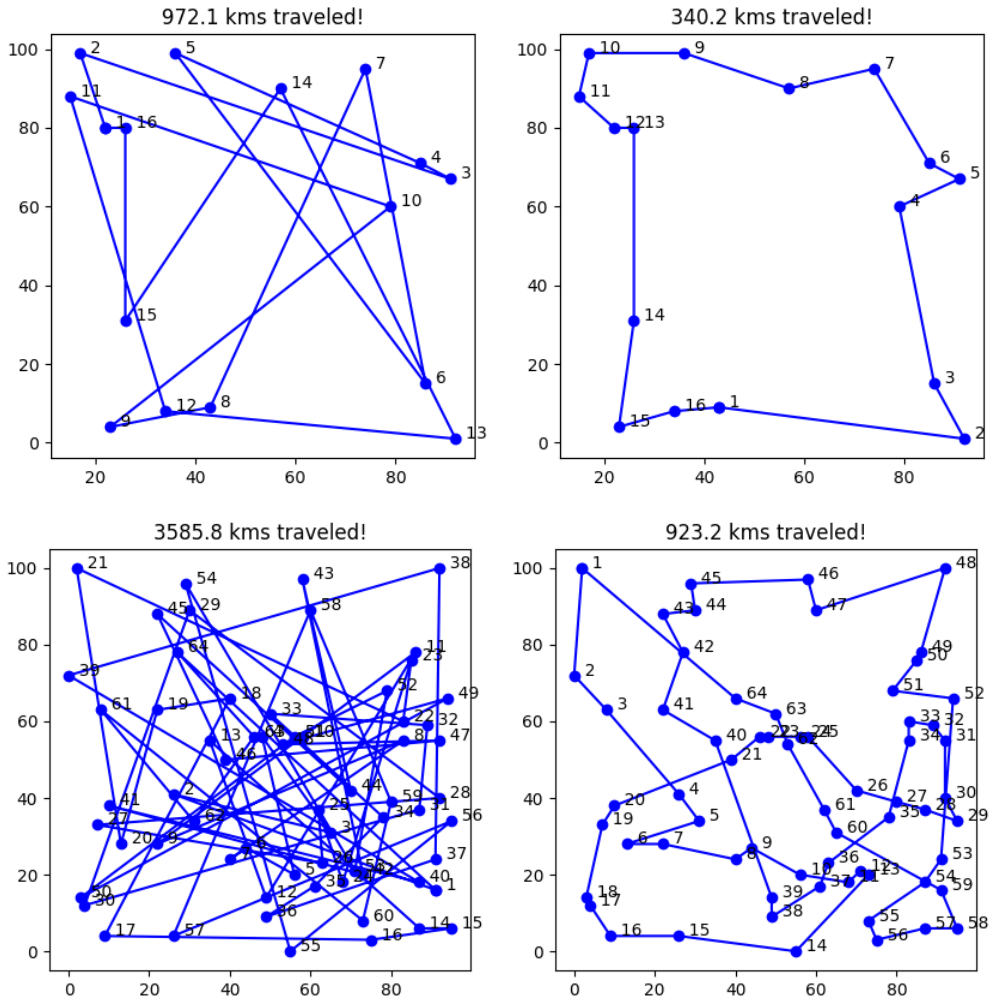
No primeiro caso, a solução parece ser a ideal. No segundo caso, com **64** cidades, a solução já não é a ideal, mas é aceitável.

Aplicando o **Random Start HillClimbing** ao problema das **NQueens**, obtêm-se os seguintes resultados



Observa-se que, tal como o **Stochastic HillClimbing**, foi capaz de resolver o problema para dimensões elevadas.

Vejamos se o **Random Start HillClimbing** é capaz de obter uma solução mais apelativa para dimensões grandes do **Travelling Salesman**.



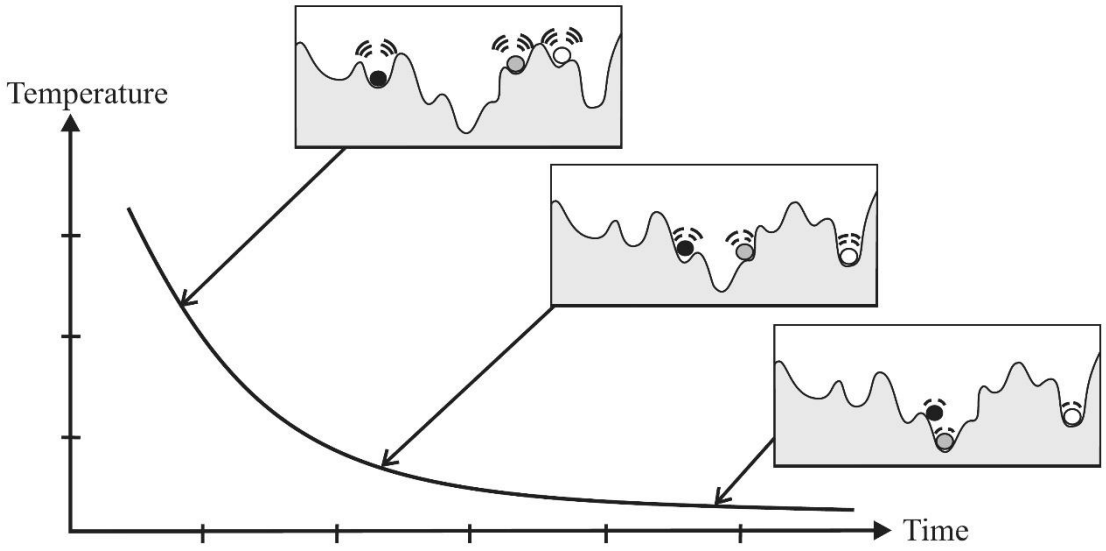
Quando o número de cidades aumenta demasiado, torna-se complicado concluir a olho nu se a solução é ideal ou não, mas parece que mesmo o **Random Start HillClimbing** não foi capaz de resolver o problema com a melhor solução possível.

4.5. Simulated Anealing

A próxima técnica a analisar será o **Simulated Aneling**.

4.5.1. Introdução Teórica

Consiste num método de têmpera simulada (analogia com a têmpera do metal, em que o aquecimento representa flexibilidade e o arrefecimento representa restrição), em que ocorre adaptação dinâmica de parâmetros do método de procura. O algoritmo é muito parecido ao **HillClimbing**, apenas não escolhe o movimento ***uphill*** (melhor movimento). Escolhe um aleatório. Se o movimento selecionado melhora a solução então é aceite. Se não melhora, existe uma probabilidade de ser aceite na mesma. Esta probabilidade decresce exponencialmente tanto com o quão mau for o movimento, como com o passar do tempo (número de iterações).



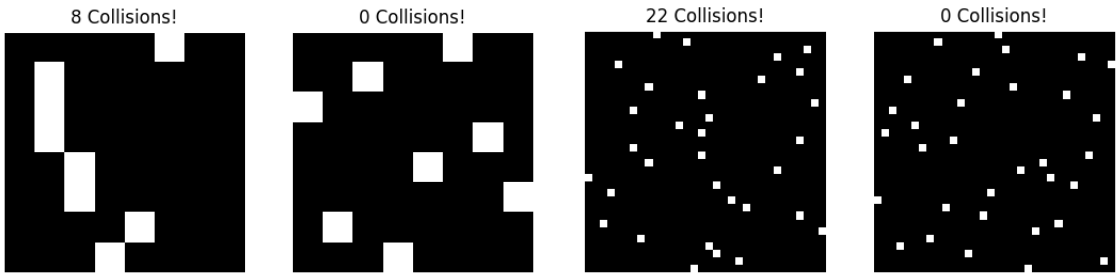
Este processo é útil para que o modelo consiga sair de mínimos locais (como ilustra a figura), dos quais o algoritmo **HillClimbing** não seria capaz de sair.

A probabilidade de aceitar o movimento incondicionalmente é:

Probabilidade(aceitar movimento) ~ 1 - exp(delta_e / T)

4.5.2. Testes e Resultados

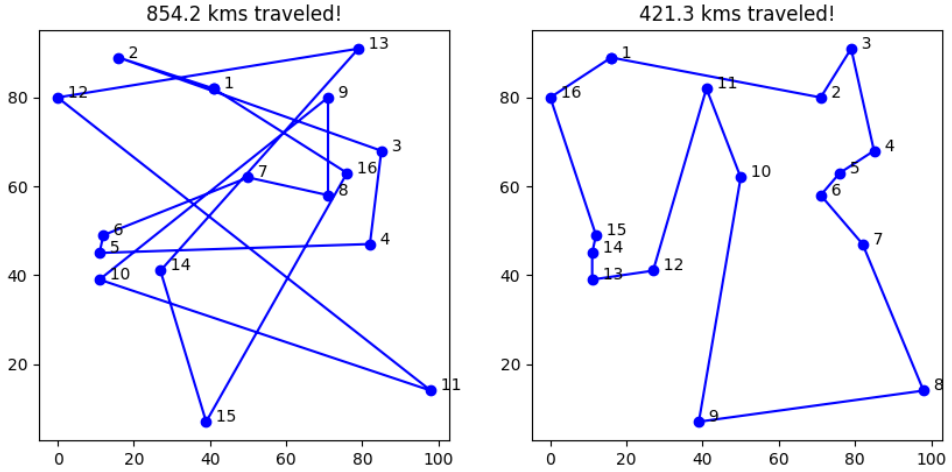
Aplicando o algoritmo de **Simulated Anealing** ao problema das **NQueens**, obtêm-se os seguintes resultados.

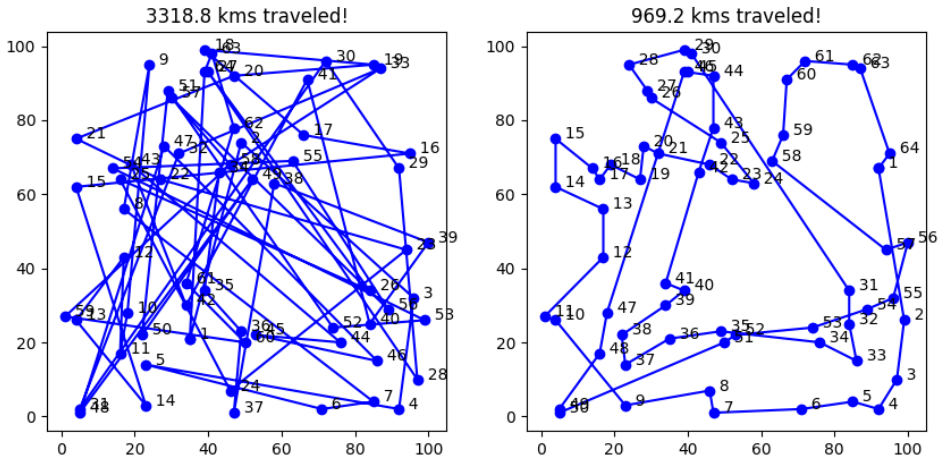


Para grandes dimensões (figura da direita), o algoritmo **Simulated Anealing** portou-se particularmente bem, e demorou muito menos tempo que o **HillClimbing**. Atingiu o valor ideal na iteração **23145**.

```
Iteration 0484 > state: [24, 31, 4, 28, 15, 21, 3, 7, 17, 10, 18, 23, 8, 27, 2, 12, 6, 1, 19, 25, 13, 15, 16, 30, 14, 7, 13, 31, 17, 27, 5, 4], value: -15.00
Iteration 0505 > state: [24, 31, 4, 20, 15, 21, 19, 7, 22, 10, 18, 23, 8, 27, 2, 12, 6, 1, 19, 25, 13, 15, 16, 30, 14, 3, 13, 31, 17, 27, 5, 4], value: -14.00
Iteration 0506 > state: [24, 31, 4, 20, 15, 21, 19, 7, 22, 10, 18, 23, 8, 27, 2, 12, 6, 1, 19, 25, 13, 15, 16, 30, 14, 3, 13, 31, 17, 28, 5, 4], value: -13.00
Iteration 0522 > state: [24, 31, 4, 20, 15, 21, 24, 7, 22, 1, 18, 23, 8, 27, 2, 12, 6, 1, 19, 25, 13, 15, 16, 30, 27, 3, 13, 31, 17, 28, 5, 4], value: -12.00
Iteration 0528 > state: [24, 31, 4, 20, 15, 21, 24, 7, 22, 1, 18, 23, 8, 10, 2, 12, 6, 1, 19, 25, 13, 15, 16, 30, 27, 3, 13, 31, 17, 28, 5, 26], value: -11.00
Iteration 0552 > state: [5, 31, 4, 20, 15, 21, 15, 7, 22, 1, 18, 23, 8, 23, 2, 12, 6, 1, 19, 25, 13, 8, 16, 30, 27, 3, 13, 31, 17, 28, 5, 26], value: -10.00
Iteration 0553 > state: [5, 31, 4, 20, 15, 21, 15, 7, 22, 1, 18, 23, 8, 23, 2, 12, 6, 1, 19, 25, 13, 4, 16, 30, 27, 3, 13, 31, 17, 28, 5, 26], value: -9.00
Iteration 0563 > state: [5, 31, 4, 20, 15, 21, 15, 7, 22, 1, 18, 23, 8, 23, 2, 12, 6, 1, 19, 25, 11, 4, 16, 30, 27, 3, 13, 31, 17, 28, 5, 26], value: -8.00
Iteration 0703 > state: [5, 27, 4, 20, 25, 21, 15, 7, 22, 24, 9, 23, 8, 26, 2, 12, 6, 1, 19, 25, 11, 14, 16, 30, 27, 7, 13, 31, 17, 26, 18, 29], value: -7.00
Iteration 0780 > state: [5, 23, 4, 20, 25, 21, 15, 7, 22, 24, 9, 23, 8, 26, 0, 12, 6, 1, 19, 27, 11, 14, 16, 30, 28, 31, 13, 3, 17, 11, 18, 29], value: -5.00
Iteration 1041 > state: [31, 23, 4, 20, 25, 21, 15, 7, 22, 24, 9, 23, 8, 2, 0, 12, 26, 1, 29, 27, 5, 14, 16, 30, 28, 31, 13, 17, 19, 11, 18, 10], value: -4.00
Iteration 1232 > state: [1, 31, 4, 20, 25, 21, 15, 7, 22, 24, 9, 23, 8, 2, 3, 12, 26, 1, 29, 27, 5, 14, 16, 30, 28, 6, 13, 17, 19, 11, 18, 10], value: -3.00
Iteration 1249 > state: [1, 31, 4, 20, 25, 21, 15, 7, 22, 24, 9, 23, 8, 2, 3, 12, 26, 1, 29, 27, 5, 14, 16, 30, 28, 6, 13, 17, 0, 11, 18, 10], value: -2.00
Iteration 3275 > state: [22, 31, 6, 13, 25, 21, 4, 7, 28, 24, 9, 23, 8, 0, 16, 12, 27, 1, 19, 2, 5, 14, 28, 30, 15, 20, 29, 17, 3, 11, 18, 10], value: -1.00
Iteration 23145 > state: [16, 8, 17, 27, 31, 13, 4, 18, 25, 11, 2, 29, 5, 1, 10, 6, 28, 22, 19, 23, 26, 15, 0, 9, 14, 24, 21, 7, 3, 12, 30, 20], value: 0.00
```

Relativamente ao problema do **Travelling Salesman**, eis os resultados obtidos.





Na primeira figura, verifica-se claramente, a olho nu, pela primeira vez, que não foi obtida a solução ideal. Contudo, para **64** cidades, o desempenho é semelhante ao **HillClimbing**.

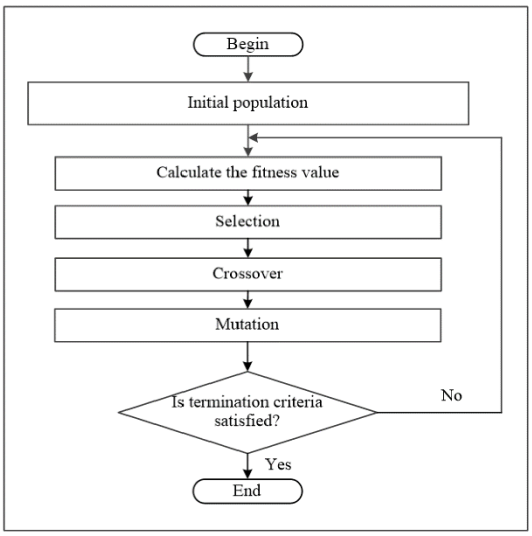
4.6. Algoritmo Genético

O último algoritmo de otimização que será estudado e implementado, é o algoritmo genético.

4.6.1. Introdução Teórica

Porquê algoritmos genéticos? Os mecanismos naturais subjacentes parecem ser bastante robustos e adequados ao suporte de mecanismos de adaptação, nomeadamente biológicos. Permitem pesquisar espaços de hipóteses complexos e compostos por elementos interdependentes, cujo impacto global na selecção da hipótese mais adequada pode ser difícil de modelar. Além disso, são facilmente paralelizáveis, permitindo por isso tirar partido de sistemas computacionais massivamente paralelos

Os algoritmos genéticos são mais complexos que os estudados anteriormente. Funciona com base em três operações fundamentais: A seleção, o ***crossover*** e a mutação.



Como indica a figura, o primeiro passo é gerar uma população inicial, isto é, um conjunto constituído por vários estados iniciais do mesmo problema (mesma dimensão e mesmos pressupostos). Cada elemento da população é designado por cromossoma, e cada elemento do cromossoma é designado por gene.

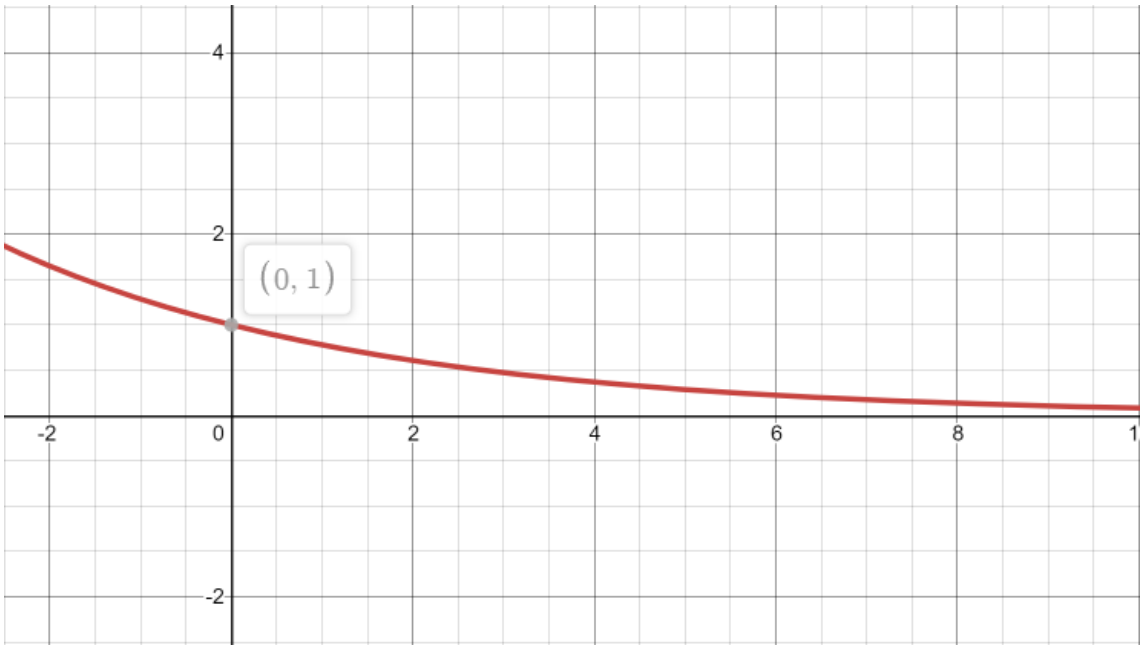
Depois, são escolhidos dois elementos da população através de um método de seleção (roleta, aleatória, estocástica, etc…) e são unidos através de um método de crossover.

Por fim, o elemento resultante, consoante uma probabilidade previamente definida, pode ou não sofrer uma mutação (alteração aleatória). Este elemento é adicionado a uma nova população. Assim que esta nova população apresentar a mesma dimensão que a população inicial, a população inicial é descartada e passa-se a trabalhar sobre a nova população (segunda geração). Este processo termina quando um determinado critério de paragem for atingido, produzindo o número de gerações que for necessário.

Para este projeto, o método de seleção a utilizar será a roleta, que consiste em escolher com mais frequência (maior probabilidade) os elementos (cromossomas) que apresentam maior fitness. A função de fitness será definida como:

fitness_function = exp(-x*(2/dimension))

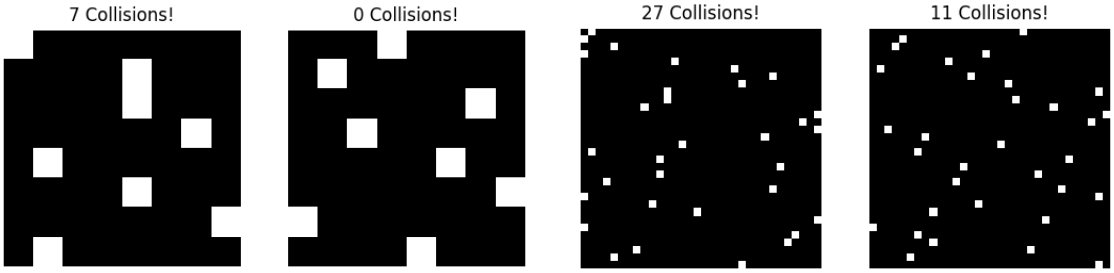
Sendo **"dimension"** uma referência da dimensão do problema. No caso do problema **NQueens**, por exemplo, com 8 rainhas, a função fitness teria o seguinte aspecto:



Isto permite que quando se obtém um número de colisões igual a **0**, o valor da função fitness é máximo (**1**), enquanto se o número de colisões for alto (**>8**), o valor da função fitness é reduzido.

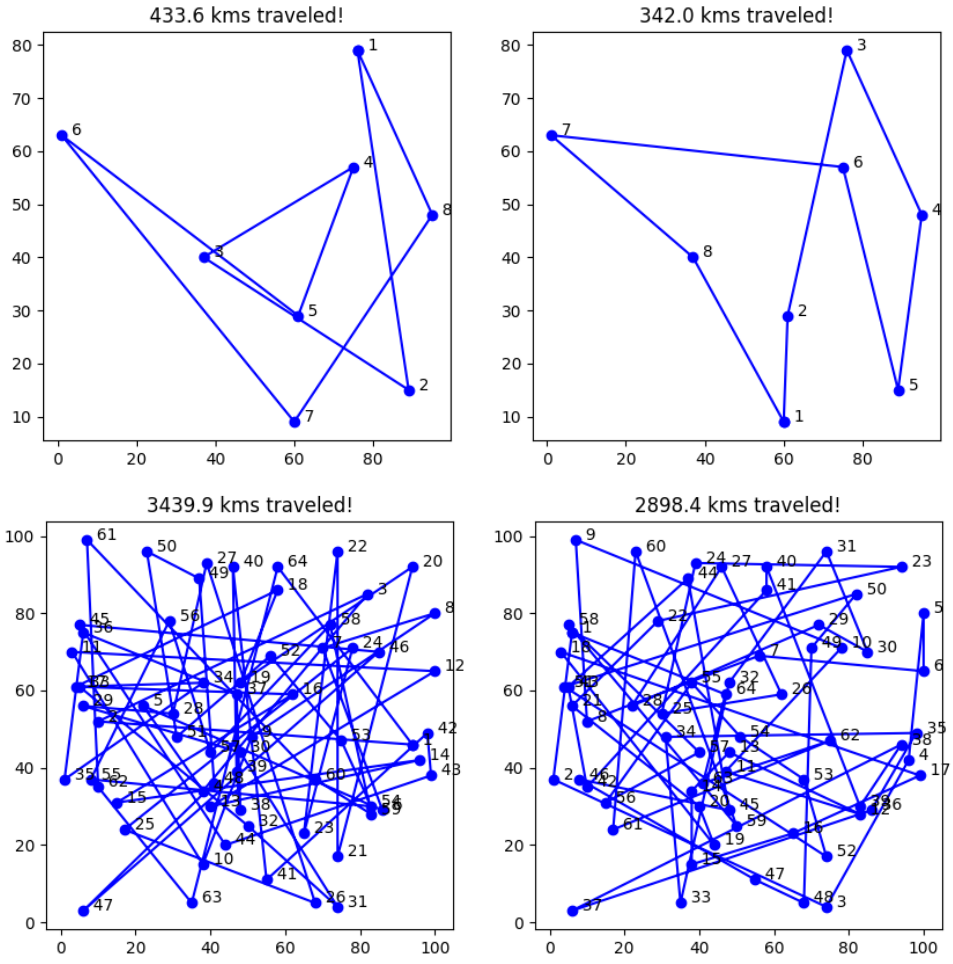
4.6.2. Testes e Resultados

Admitindo um máximo de gerações igual a **48** e um número de cromossomas por população igual a **100**, estes são os resultados obtidos quando se aplica o algoritmo genético ao problema **NQueens**.



Verifica-se que para um problema com grandes dimensões, o algoritmo genético não tem a capacidade de o resolver, demorando imenso tempo, e ficando ainda bastante longe da solução ideal.

Vejamos como se sai, quando aplicado ao problema do **Travelling Salesman**.

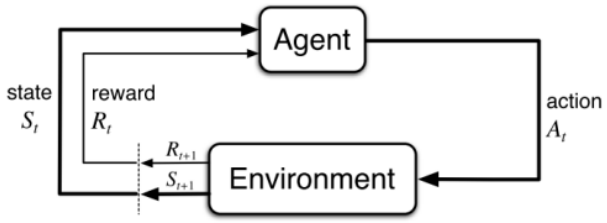


Nem para 8 cidades foi capaz de encontrar a solução ideal, e muito menos para 64 cidades. O algoritmo genético implementado tem um péssimo desempenho para problemas com grandes dimensões.

5. Aprendizagem por Reforço e Raciocínio Automático para Planeamento

5.1. Introdução Teórica

A aprendizagem por reforço caracteriza-se por uma aprendizagem comportamental, no sentido em que tira partido de iterações com o ambiente para obter informação. De forma muito abstrata, diz-se que aprendizagem por reforço é aprender o que fazer (relacionar ações com situações) de modo a maximizar um determinado sinal de recompensa numérico. Este sinal de recompensa (***Reward***) informará o agente (***Agent***) acerca da boa/má decisão (***Action***) realizada. De cada ação, resulta um estado (***state***) que representa o progresso. Observe-se a figura seguinte, que ilustra esse mecanismo.



Então, existem três grandes conceitos que se relacionam para fazer este mecanismo funcionar: A Ação, o Estado e o Reforço (***Reward***, que pode ser positiva ou negativa, isto é, um ganho ou uma perda).

Existem dois conceitos que serão também importantes (principalmente para o algoritmo **DynaQ**):

- **Exploração (*explore*):** Almeja explorar todos os estados possíveis, experimentando todas as ações. Permite a obtenção de experiência (principalmente na fase inicial, em que o agente ainda está a apalpar terreno, e não possui qualquer experiência).
- **Aproveitamento (*exploit*):** Utiliza o conhecimento resultante da exploração (aprendizagem) para obter o máximo de recompensa. Restringe-se às ações que se conhece serem favoráveis.

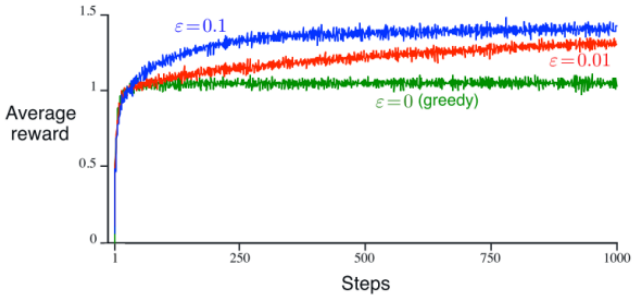
Surge, por isso, um dilema. Quando é que se aprendeu o suficiente para começar a aplicar o que se aprendeu? É aí que entram técnicas com o **EGreedy**, que permitem obter esse equilíbrio. O ideal seria que se no início se recorra mais à exploração do que ao aproveitamento, para gerar experiência, visto nas primeiras iterações ela é pouca ou nula. Contudo, à medida que se obtém experiência, o recurso ao aproveitamento deverá ser cada vez mais frequente.

A estratégia de seleção de ação **EGreedy** não permite esse balanceamento, visto que se baseia na aleatoriedade, no sentido em que é um parâmetro épsilon (**ε**) que define a probabilidade de se escolher a ação de exploração. Normalmente escolhe-se um épsilon entre **0** e **0.1**, ou seja, na maior parte das iterações a ação selecionada será o aproveitamento, quer tenha experiência suficiente ou não.

No gráfico da página seguinte, observa-se que, para um épsilon igual a **0**, a estabilização do valor médio da recompensa ocorre muito rapidamente, logo nas primeiras iterações. Isto acontece porque a probabilidade de selecionar a ação de exploração é nula, e por consequente, o modelo avança com base apenas no aproveitamento, sem nunca obter experiência através da exploração.

No caso de um épsilon igual a **0.1**, o valor médio da recompensa é reduzido, visto que o modelo se encontra numa fase de exploração e de obtenção de experiência.

À medida que se avança nas iterações (***steps***), o valor médio da recompensa aumenta, pois o modelo consegue tomar decisões acertadas de forma mais frequente.



Existem dois tipos de aprendizagem. Com política de seleção de ação única (***on-policy***) e com políticas e seleção de ação diferenciadas (***off-policy***).

- **On-policy:** Utilização da mesma política de seleção de ação para comportamento e para propagação de valor e exploração de todas as ações (por exemplo, estratégia **EGreedy**).
- **Off-policy:** Utilização de diferentes políticas de selecção de ação para comportamento e para propagação de valor (algoritmo **QLearning**).

Os algoritmos de aprendizagem com memória de experiência são algoritmos em que as experiências do agente são memorizadas numa memória de experiência (***replay memory***) que é utilizada para simular a experiência real. Almeja acelerar o processo de aprendizagem. A seleção de amostras é aleatória ou por critério de relevância. Nos capítulos seguintes abordar-se-ão técnicas que tiram partido do modelo de um mundo. Esse modelo terá de ser capaz de representar as características relevantes do domínio do problema que se pretende resolver, e deverá conter informação acerca da estrutura (estado), dinâmica (transição de estado) e valor (recompensa). Este modelo será utilizado para simular experiência, deverá suportar planeamento e deverá manter-se atualizado e consistente com a experiência.

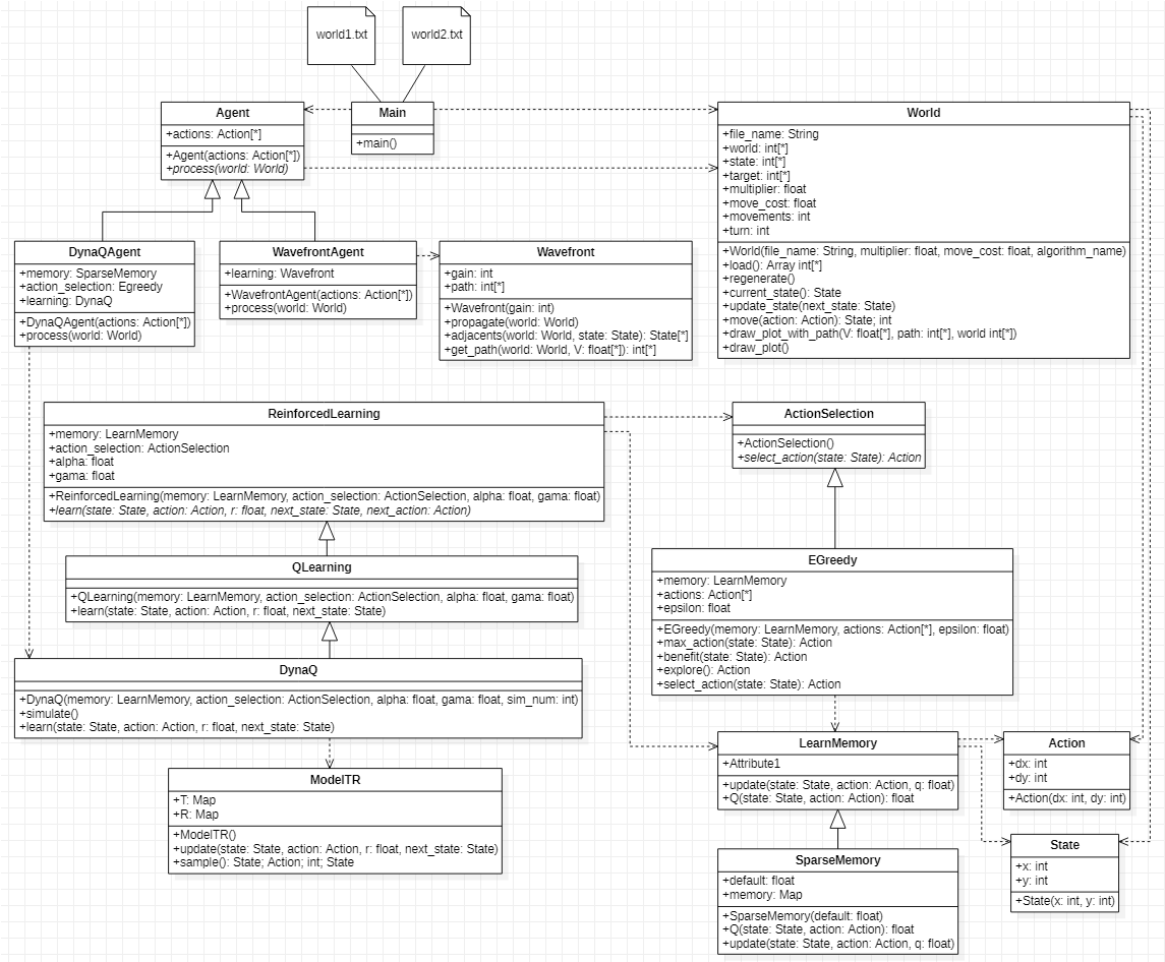
5.2. Problema de Navegação até ao Alvo

Este problema procurará levar o quadrado verde (o quadrado mais à esquerda) até ao quadrado amarelo (mais à esquerda), utilizando técnicas de aprendizagem por reforço e raciocínio automático para planeamento.

Obviamente, a capacidade do agente de realizar esse feito deverá ser independente do ambiente, isto é do local onde inicia, da localização do alvo (amarelo), dos obstáculos (desde que haja um caminho possível) e até da dimensão do mapa. Na figura está exemplificado um dos ambientes que serão utilizados na implementação deste problema. Note-se que os obstáculos estão pintados a roxo.

5.3. Implementação

Como se pretende explorar diferentes algoritmos e diferentes ambientes, dever-se-á abordar este problema de forma metódica e estruturada. Assim, foi desenvolvido o seguinte diagrama de classes:



Note-se que o programa principal (**Main**) estará abstraído da maior parte das classes, tendo apenas de saber que mundo (**World**) inicializar e que agente (**Agent**) utilizar. Os dois algoritmos a explorar são o **DynaQ** e o **Wavefront**. Serão devidamente estudados, analisados e testados em capítulos subsequentes.

5.4. Algoritmo Wavefront

Explore-se agora o algoritmo **Wavefront**, um algoritmo de raciocínio automático para planeamento.

5.4.1. Introdução Teórica

O algoritmo Wavefront baseia-se na expansão de nós com o tipo de procura em largura. Começando pelo estado alvo (***target***) procuram-se os estados adjacentes e, a cada um, atribui-se um valor numérico que será cada vez mais reduzido à medida que o algoritmo de afasta do estado alvo. Ou seja, os primeiros valores (perto do estado alvo) apresentarão valores relativamente elevados, enquanto os que estão no lado contrário do mundo receberão valores bastanta mais reduzidos.

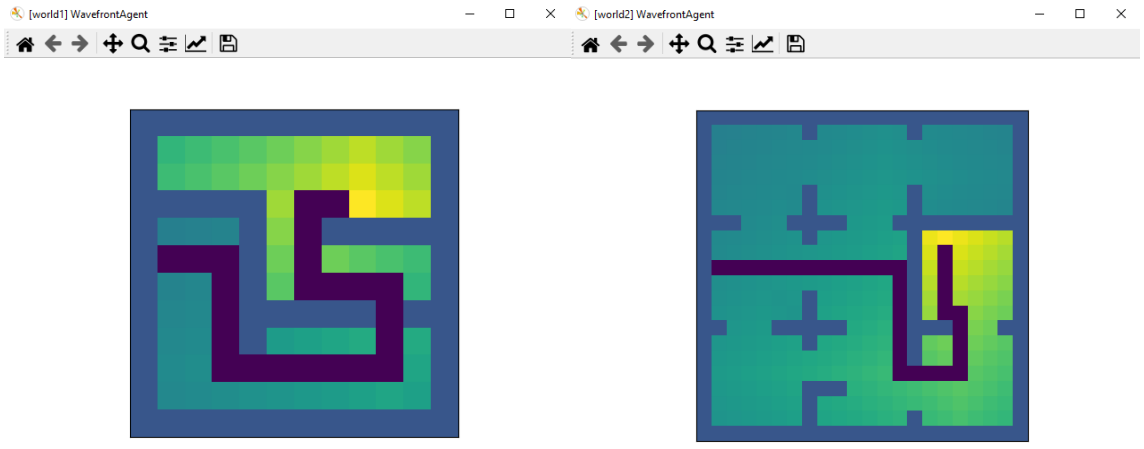
Este processo resulta numa representação do mundo, em que cada estado apresenta um valor. Com base nessa representação torna-se trivial a descoberta de um caminho eficaz até ao estado alvo, partindo de qualquer local do mundo.

A descoberta desse caminho poderia ser feita através de um algoritmo do tipo hill-climbing, que procuraria o melhor vizinho de cada estado (com maior valor numérico). Contudo, neste caso, tirar-se-á partido do método implementado para procurar estados adjacentes. Obtendo os estados adjacentes ao estado corrente, consegue-se obter o melhor vizinho através de uma simples comparação.

Uma grande vantagem deste algoritmo é que, visto que se trata de uma procura por frente-de-onda, é garantido (sempre que haja pelo menos um caminho possível) que o agente chega ao destino. Contudo, apresenta uma desvantagem crucial. O mundo fica representado em memória, e no caso de se tratar de um mapa com grandes dimensões, isso pode ter custos elevados. Além disso, sempre que o alvo mudar de local, o processo tem de ser repetido para gerar novos valores numéricos.

5.4.2. Testes e Resultados

As seguitnes figuras representam o caminho decidido pelo algoritmo Wavefront, em que a figura da esquerda ilustra o mundo **"world1"** e a figura da direita ilustra o mundo **"world2"**. Note-se também que, para valores mais elevados (para estados mais próximos do alvo), a cor torna-se mais clara (um verde mais fluorescente), criando um efeito de gradiente.



5.5. Algoritmo DynaQ

Por último, explorar-se-á o algoritmo **DynaQ**, um algoritmo de aprendizagem por reforço.

5.5.1. Introdução Teórica

O algoritmo **DynaQ** é um algoritmo de aprendizagem por reforço que tira partido de uma memória esparsa para guardar pares estado-ação (***state and action***), e utiliza um método de seleção de ação baseado na estratégia **Egreedy**.

O valor da ação guardado na memória é atualizado consoante a ação que maximiza o valor de Q(s, a) do estado seguinte (sendo **s** o estado e **a** a ação), consoante o reforço, e consoante o Q do estado corrente. Este mecanismo permite que o agente escolha a melhor decisão a longo prazo, isto é, se existe um movimento com custo elevado, mas que evita um percurso maior o agente deverá ser capaz de concluir que o movimento de custo maior é o que mais compensa a longo prazo.

O parâmetro ***alpha***, na aprendizagem por reforço (classe **ReinforcedLearning**) simboliza o fator de aprendizagem que indica ao agente o quão ele deverá ter em conta esse tipo de situações. Será colocado a **0.7**.

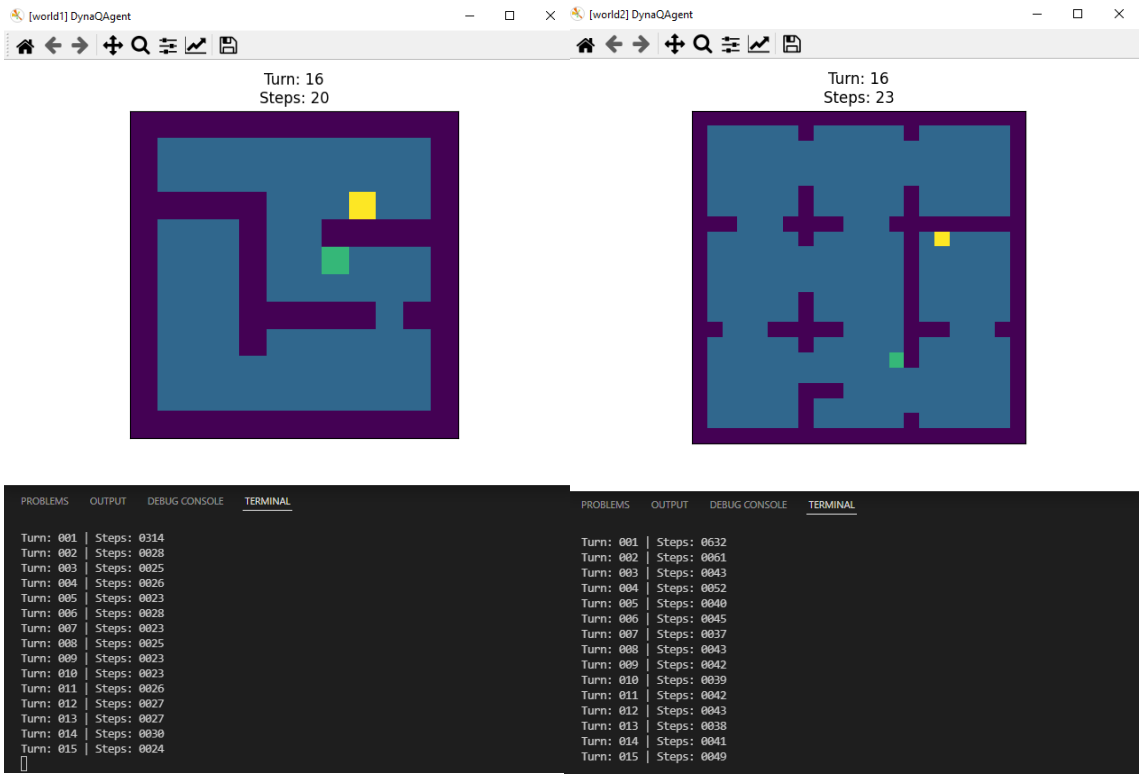
O parâmetro ***gama***, também na aprendizagem por reforço, simboliza o fator de desconto. Será colocado a **0.95**.

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow \varepsilon$-greedy(S,Q)
- (c) Take action A; observe resultant reward, R, and state, S'
- (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
- (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
- (f) Loop repeat n times:
  - $S \leftarrow$ random previously observed state
  - $A \leftarrow$ random action previously taken in S
  - $R, S' \leftarrow Model(S,A)$
  - $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

5.5.2. Testes e Resultados

Ao executar o agente, obtêm-se os seguintes resultados:



Observa-se que na primeira iteração, existe sempre um maior número de passos realizados por parte do agente. Isto deve-se ao facto de, nessa altura, ele não ter qualquer experiência. Contudo, a partir da segunda iteração, o desempenho do agente melhora significativamente, e são necessários muito menos passos para chegar ao alvo, revelando que o agente aprendeu e apresenta agora experiência, a partir da qual consegue tomar decisões mais acertadas.

Observa-se também que, para mundos com dimensões maiores, é necessário mais passos porque o processo de exploração é mais demorado.

5.6. Algoritmo RTAA*

Este algoritmo não foi implementado, pelo não se poderão realizar testes. Contudo, é uma algoritmo que pratica um tipo de procura informada, ou seja, tendo conhecimento do domínio do problema (ambiente), obtendo o custo do percurso desde o nó inicial ao nó alvo, e estimando o custo de atingir o alvo a partir do nó atual (função heurística), é possível calcular uma estimativa do custo do percurso total.

Tira partido de uma procura melhor-primeiro, isto é, combina uma procura de custo uniforme, **g(n)**, com uma procura sôfrega, **h(n)**.

Procura A* = f(n) = g(n) + h(n)

6. Aplicação de AI à Detecção de Acordes

No **capítulo 3.4**, foi estudada uma forma de detectar acordes musicais com base em dados de entrada **MIDI**, e com recurso a uma rede neural. O nível de complexidade desse método é muitíssimo reduzido, pelo que:

- Recorre a uma representação digital de informação musical (**MIDI**), quando a grande maioria da música é apresentada ao público em formato analógico;
- Identifica acordes muito simples (apenas os acordes padrão maiores e menores), quando existem muitos outros acordes e claro, outros aspetos que também seriam interessantes de concluir acerca de uma música;
- Para gerar o ***target data***, recorre a uma método que já é capaz, por si só, de concluir, dada uma oitava, qual o acorde que está a ser reproduzido;
- Assume que apenas existe uma oitava, quando existem muitas outras frequências que, apesar de não poderem fugir às **12** notas fundamentais, produzem efeitos sonoros muito díspares.

Antes de abordar a matéria de inteligência artificial e sistemas cognitivos no âmbito deste problema, talvez seja importante abordar algumas temáticas igualmente importantes.

6.1. Teoria Musical

O problema em questão envolve teoria musical, no sentido em que se pretende exatamente concluir aspetos acerca de uma peça musical, com recurso à tecnologia de processamento digital de sinal e inteligência artificial. O conceito de teoria musical não é algo como um teorema matemático, que procura definir algo como certo e definitivo. É mais como umas cábulas que um aluno do 6º ano escreve nas costas da mão antes de realizar um teste de história. A melhor definição seria uma tentativa de descrever, por palavras, os elementos musicais e sua relação. Contudo, é importante que se perceba que a música é algo que é sentido e ouvido, pelo que não tem de respeitar quaisquer regras (como qualquer forma de arte). Os conceitos e terminologia associados à teoria musical ajudam sobretudo na comunicação e na percepção visual dos componentes de uma música, mas não é algo que tem de ser respeitado.

Por exemplo, as músicas populares (***pop***) muitas vezes seguem uma fórmula que constitui uma sequência de acordes padrão que, na terminologia musical, se denominam por acordes de grau **1**, **4**, **5** e **6**. São acordes que estimulam a audição por soarem bem juntos, e por isso são bastante comuns. A teoria musical explica que esses acordes pertencem à escala pentatónica de um determinado tom.

O tom de uma música é, na sua essência, e da forma mais abstrata possível, designar uma música inteira por uma única nota. Por exemplo dizer: "Esta música está no tom de Dó", significa que o acorde de grau **1** é Dó maior, o de grau **2** é Ré menor e por aí fora. Em particular, a escala diatónica de Dó constitui as **7** notas brancas de uma oitava, pelo que a teoria musical incorreria no risco de admitir que essas são as notas permitidas neste tom. E em alguns casos é verdade. As únicas notas tocadas são realmente essas. Mas existem exceções (como já foi abordado, não se estabelecem regras na música).

Conclui-se que a teoria musical seria, sem dúvida, algo que um sistema de inteligência artificial teria em conta neste contexto, mas a composição musical é, na maior parte das vezes, orientada ao que uma determinada sucessão de notas nos faz sentir, e não ao que está escrito no papel. Estes conceitos de acorde, tom, escala diatónica, escala pentatônica, etc… seriam importantes para o modelo de inteligência artificial, pois permitem que o sistema encontre e reconheça padrões.