VectorRace

Inteligência Artificial

Filipe Felício
A85983

Luís Pereira
A96681

Rui Oliveira
A95254

Tiago Pereira
A95104

Grupo 30

Departamento de Informática
Universidade do Minho

novembro 2022

1 Introdução

O presente relatório é referente à primeira fase do trabalho prático da unidade curricular de Inteligência Artificial da Licenciatura em Engenharia Informática da Escola de Engenharia da Universidade do Minho, para o ano letivo de 2022/2023. O projeto foi desenvolvido por Filipe Felício (A85983), Luís Pereira (A96681), Rui Oliveira (A95254) e Tiago Pereira (A95104).

O trabalho prático consiste na implementação de algoritmos de procura no contexto do jogo VectorRace, que consiste num jogo de simulação simplificada de corridas de carros.

Ao longo do relatório, o problema será formalizado como um problema de procura em grafos; será explicada a forma como se geram circuitos e se convertem para grafos, e explicar os diferentes algoritmos utilizados para resolver o problema.

Finalmente, serão detalhadas funcionalidades adicionais que o grupo considera um fator diferenciador relativamente a outros trabalhos.

2 Formalização do Problema

Consideremos um circuito como sendo uma grelha bidimensional com l linhas e c colunas. Um carro começa numa posição inicial (x_i, y_i)[1], e tem como objetivo chegar a uma ou mais posições finais (x_f, y_f).

Cada célula pode ser:

- Fora de pista
- Pista
- Posição Inicial
- Posição Final

Em cada momento da corrida, o carro tem uma dada velocidade v, que pode ser decomposta nas suas componentes (v_x, v_y). A qualquer momento, pode escolher acelerar ou desacelerar uma unidade em cada direção. Também é possível o carro escolher não acelerar, mantendo a sua velocidade atual. Sempre que o carro sai de pista, volta à sua posição anterior com velocidade nula.

Deste modo, é possível formular este problema com um problema de procura do seguinte modo. Consideremos o estado do problema como sendo o conjunto da posição do carro e a sua velocidade. Formalmente, podemos considerar o estado como o par (p, v), que pode ser decomposto em $((p_x, p_y), (v_x, v_y))$.

O estado inicial corresponde a $(p_i, (0, 0))$, em que p_i é a posição inicial do circuito, e os estados finais correspondem a (p_f, v_f), em que p_f é uma das posições finais do circuito e v_f pode ser qualquer valor de velocidade.

Num dado estado $((p_x, p_y), (v_x, v_y))$, os estados seguintes possíveis, dependendo da jogada do carro, são:

1. $((p_x + v_x + 1, p_y + v_y + 1), (v_x + 1, v_y + 1))$
2. $((p_x + v_x + 1, p_y + v_y), (v_x + 1, v_y))$
3. $((p_x + v_x, p_y + v_y + 1), (v_x, v_y + 1))$
4. $((p_x + v_x - 1, p_y + v_y + 1), (v_x + 1, v_y + 1))$
5. $((p_x + v_x + 1, p_y + v_y - 1), (v_x + 1, v_y + 1))$
6. $((p_x + v_x - 1, p_y + v_y - 1), (v_x + 1, v_y + 1))$
7. $((p_x + v_x - 1, p_y + v_y), (v_x + 1, v_y + 1))$
8. $((p_x + v_x, p_y + v_y - 1), (v_x + 1, v_y + 1))$
9. $((p_x + v_x, p_y + v_y), (v_x, v_y))$

[1] A coordenada (x, y) significa que o carro se encontra na $x + 1$-ésima coluna e na $y + 1$-ésima linha

10. $((p_x, p_y), (0, 0))$, em caso de colisão com parede

O custo associado a cada movimento é 1, se não houver nenhuma colisão com um obstáculo; e 25 caso contrário. Pretende-se, naturalmente, minimizar o custo associado a deslocar-se da posição inicial à final. A forma como são detetadas colisões com paredes é discutida de seguida.

2.1 Deteção de Colisões

De forma a impedir que o carro se desloque através de paredes é necessário definir mais rigorosamente o que se quer dizer com *através de paredes*, visto que o carro se desloca vários quadrados na mesma jogada.

Assumamos que um carro se encontra no estado $(p, v) = ((0, 3), (5, -3))$ e escolhe não acelerar $(a = (0, 0))$. O estado seguinte, como explicado na secção anterior, será de $((5, 0), (5, -3))$ caso não seja detetada uma colisão e $((0, 3), (0, 0))$ se for detetada. Se admitirmos que a pista em redor do carro tem a seguinte disposição de paredes, qual deve ser o próximo estado do carro?

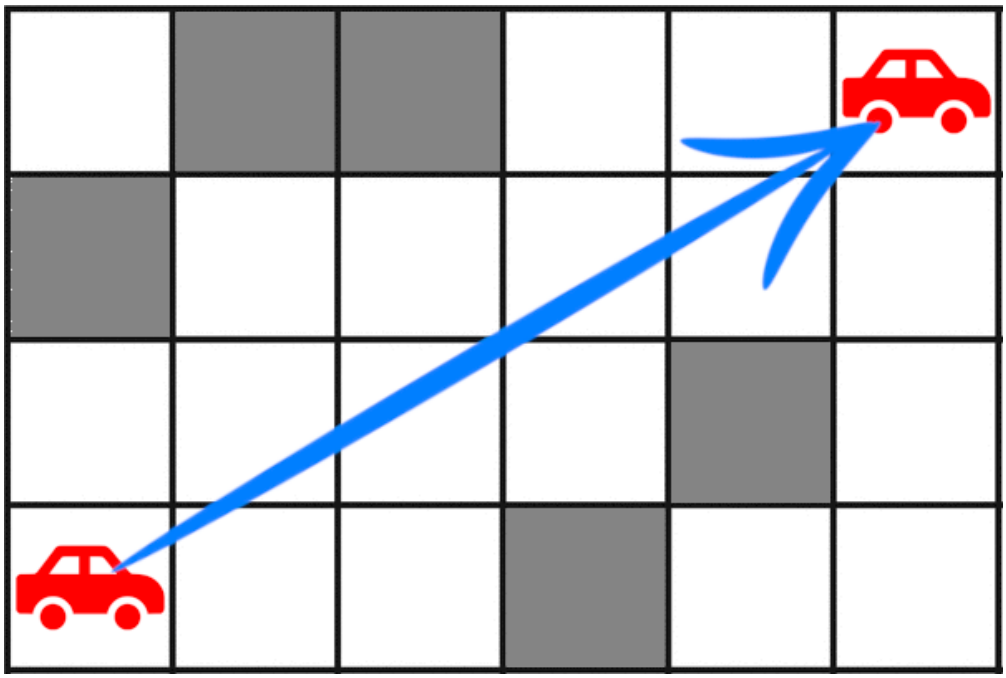

Figura 1: Exemplo de movimentação

A forma como o grupo solucionou este problema foi modelar o carro como um quadrado, que desloca em linha reta da posição $(0, 3)$ até à posição $(5, 0)$. Consideramos que ocorreu uma colisão se em qualquer ponto dessa trajetória, a interseção deste quadrado com as paredes da pista tiver área não nula. Esta formulação é equivalente a considerar apenas três segmentos de reta entre os dois quadrados, duas que ligam os cantos dos quadrados exteriores à translação e uma que liga os centros dos quadrados (numa movimentação horizontal ou vertical podemos calcular apenas a reta do interior). Se alguma destas retas intersetar o interior de alguma parede, consideramos que ocorreu uma colisão.

Entrando em mais detalhe, de forma a determinar os quadrados que um segmento de reta interseta consideramos a sua interseção com as retas horizontais e verticais que compõem a grelha. Para cada interseção verificamos os dois quadrados adjacentes. Se algum deles for uma parede, ocorreu uma colisão.

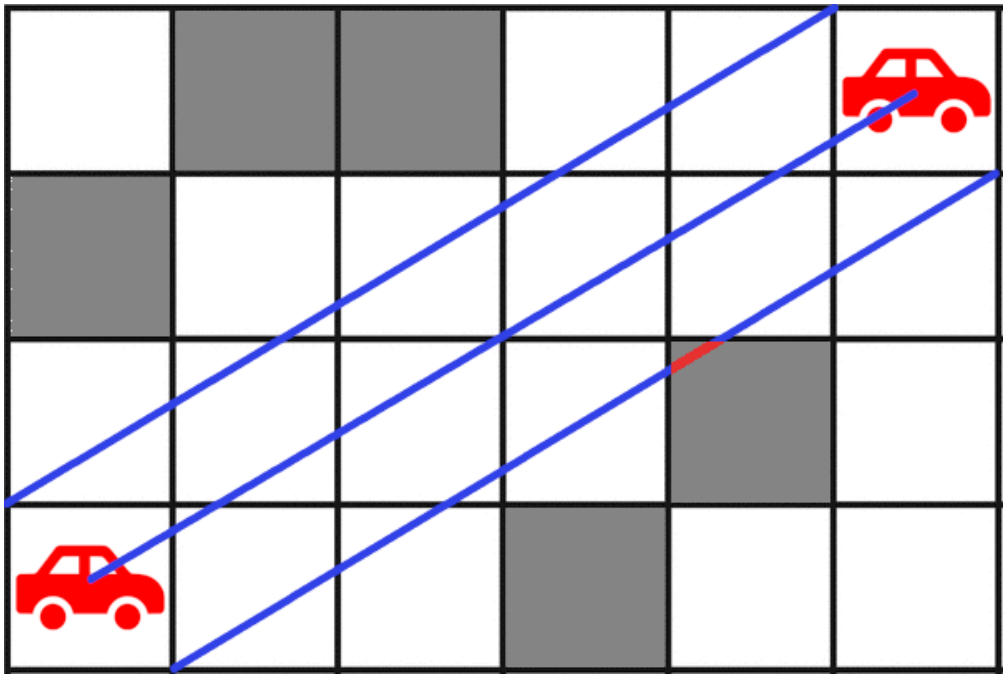

Figura 2: Exemplo de movimentação

3 Circuitos

Como mencionado anteriormente, o circuito pode ser expresso como uma grelha bidimensional de l linhas e c colunas. Cada símbolo descreve uma célula da grelha com a correspondência expressa na tabela 1.

Símbolo	Significado
X	Obstáculo
-	Pista
P	Posição Inicial
F	Posição Final

Tabela 1: Correspondência entre Símbolos e o seu significado nos circuitos

Um exemplo de um circuito é o seguinte.

X X X X X X X X X X
X X – – – X X – – X
X – – – – – – – – F
X P – – X X X – – F
X – – – – – – – – F
X X X X – – – – X X
X X X X X X X X X X

3.1 Geração

Um dos aspetos diferenciadores deste projeto é a forma utilizada para gerar circuitos. Em vez de os gerar de forma manual, criando e povoando o ficheiro sem qualquer tipo de automatização; o grupo criou um sistema que permite gerar circuitos bastante mais complexos de forma automática a partir de uma imagem.

Na imagem, todos os pixeis brancos correspondem a obstáculos; os pretos a pista; os vermelhos a posições finais, e os verdes a posições iniciais.

Figura 3: Circuito em imagem

Mais concretamente, o programa desenvolvido, ao arrancar, varre a diretoria gui/ circuits à procura de imagens. Depois dá a possibilidade ao utilizador de escolher que circuito pretende utilizar na corrida.

Após essa seleção, o programa lê o ficheiro e gera o circuito. Em primeiro lugar, modifica a imagem carregada em memória para ter um tamanho de 100x50 pixeis.

A partir desta fase o sistema identifica todos os pixeis vermelhos RGB(255,0,0) ± (50,30,30),verdes RGB(0,255,0) ± (30,50,30) e pretos RGB(0,0,0) ± (100,100,100). Daí, surgem mapas com a cor de cada pixel em cada coordenada conseguindo gerar em simultâneo uma matriz com a representação em texto de cada coordenada do circuito e uma nova imagem com as cores finais a mostrar.

Figura 4: Circuito final

3.2 Conversão para Grafos

A conversão do circuito para um grafo consiste, essencialmente, numa travessia em profundidade do grafo implícito definido pela grelha e pelas regras definidas anteriormente.

Mais concretamente, inicia-se uma travessia a partir de cada posição inicial começando com velocidade nula, inserindo todos os estados visitados no grafo. Um estado é definido pela posição (x,y) e velocidade (vx,vy) do carro. Para cada estado visitado, calculam-se todos os estados adjacentes a partir das regras definidas anteriormente, adicionando esses nodos ao grafo (se ainda não estiverem presentes), e adicionando um arco que os liga.

Este sistema de conversão é mais vantajoso do que o mais natural, que seria, no ótica do grupo, considerar um grafo sem velocidades, em que a travessia seria apenas célula a célula; uma vez que permite desenvolver os

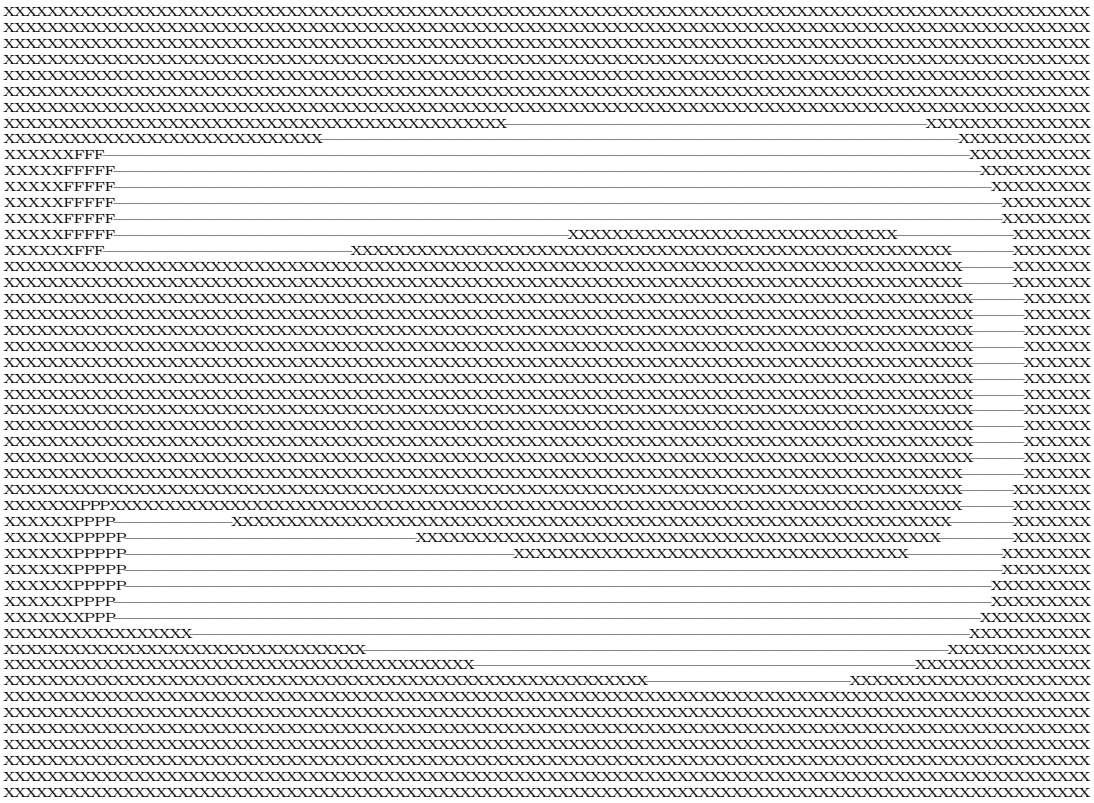

Tabela 2: Circuito final em texto.

algoritmos de forma bastante linear, sem necessidade de grandes adaptações às particularidades do problema. De qualquer dos modos, é necessariamente mais difícil de implementar do que a versão mais óbvia.

3.3 Cálculo de Heurística

Em algumas das estratégias definidas na próxima secção é utilizado o conceito de heurística de utilidade de um nodo. Neste projeto, a utilidade é calculada a partir da posição dos nodos de chegada, com a assunção que o caminho em linha reta entre o nodo inicial e o nodo de chegada mais próximo não tem quaisquer obstáculos. É maior a utilidade de um nodo quanto menor for o número de movimentos necessários para percorrer esse caminho. Qualquer heurística mais detalhada implicaria utilizar informação sobre o circuito, o que o grupo considerou demasiado lento computacionalmente para o cálculo da heurística.

4 Estratégias de Procura

Ao iniciar o programa, para além de estarem disponíveis vários circuitos nos quais simular a corrida, estão também presentes várias estratégias de procura. Estas estratégias definem o comportamento dos carros e são descritas em pormenor nas secções seguintes.

Uma análise empírica da aplicação destas estratégias encontra-se detalhada na secção 7.2.

4.1 Procura em Profundidade

Esta estratégia consiste numa travessia em profundidade até encontrar os nodos da saída, ou seja, para explorar um nodo explora-se recursivamente cada um dos nodos adjacentes (ignorando nodos já explorados para evitar ciclos infinitos, uma vez que o grafo sobre o qual estamos a procurar não é necessariamente uma

árvore). Isto significa que o caminho encontrado até à saída não será o mais curto nem o menos custoso, mas antes aquele que, por acaso da geração do grafo, seja explorado em primeiro lugar.

4.2 Procura em Largura

Nesta estratégia, explorar um nodo consiste em explorar todos os nodos adjacentes, explorar os nodos adjacentes a esses, e assim por diante, até atingir um nodo de saída. Formalmente, todos os nodos a uma distância n do nodo inicial são explorados antes de explorar os nodos a uma distância $n + 1$. Isto garante que o caminho resultante terá o menor comprimento possível, mas não dá qualquer garantia quanto ao custo desse caminho, uma vez que colidir com uma parede tem um custo de 25.

4.3 Gulosa

Como vemos pelo nome desta estratégia, em cada nodo explorado é escolhido para ser explorado em seguida o nodo adjacente mais apelativo, ou seja, com maior heurística de utilidade, ignorando os restantes. Isto leva a que esta estratégia, ao contrário das restantes, não consiga dar resposta a circuitos possíveis, nomeadamente a circuitos em que é necessário o jogador deslocar-se na direção oposta à chegada durante parte do circuito. Nessas situações, a estratégia gulosa vê-se presa num ciclo, nunca terminando.

4.4 A*

Tal como a estratégia gulosa, esta estratégia utiliza a heurística de utilidade. No entanto, ao contrário da estratégia gulosa, A* não descarta os caminhos não tomados, podendo "voltar atrás"caso o caminho considerado não ser viável. É também tido em conta o custo do caminho percorrido até ao nodo, de forma a minimizar o custo do caminho encontrado.

No caso geral, com uma qualquer função heurística, o algoritmo A* não faz qualquer garantia quanto ao custo da solução encontrada. No entanto, usando uma heurística admissível (que nunca sobrestima o custo de um caminho até ao nodo de chegada), a estratégia retorna garantidamente um caminho ótimo. Como a função heurística por definição define-se como o custo do caminho teórico mais curto (ver secção 3.3), o caminho encontrado é garantidamente ótimo.

4.5 Procura em Profundidade (Iterativa)

Nesta estratégia são realizadas várias procuras em profundidade com um limite de profundidade cada vez maior, até um nodo de chegada ser encontrado. De certa forma, esta estratégia combina as vantagens das estratégias de procura em profundidade e largura, mantendo a propriedade de visitar os nodos a distância n antes de visitar os nodos a distância $n + 1$, que garante que uma solução com comprimento mínimo é encontrada, enquanto que mantém a simplicidade e facilidade de implementação de uma pesquisa em profundidade (poupando memória visto não ser necessário armazenar todos os nodos à espera de serem visitados).

Como desvantagem, vários nodos, especialmente os mais próximos do nodo inicial, são visitados várias vezes antes da solução ser encontrada.

5 Funcionalidade Adicional: Interface Gráfica

A interface gráfica está dividida em 3 sub-menus:

5.1 Menu Principal

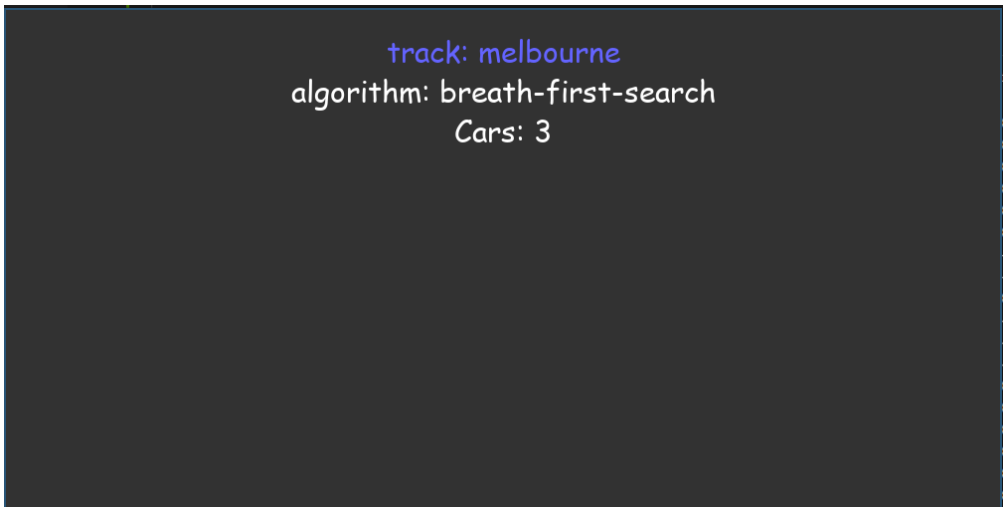

Figura 5: Menu Principal

O menu principal tem 3 linhas, aparecendo a azul a linha atualmente selecionada:

- Track: representa qual dos ficheiros contidos em gui/circuits deve ser utilizado para converter para grafo a usar na simulação.
- Algorithm: representa qual dos algoritmos disponíveis vai ser utilizado para fazer a pesquisa no grafo durante a simulação
- Cars: representa o numero de carros que vão ser simulados durante a simulação. Cada carro é simulado independentemente.

5.2 Menu Simulação

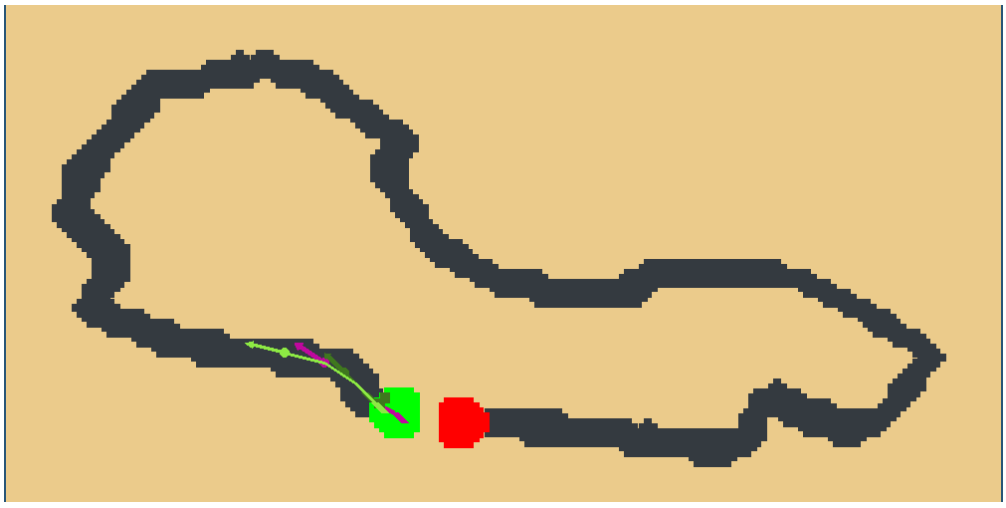

Figura 6: Menu Simulação

Neste menu é possível ver passo a passo o percurso que cada carro faz no circuito. Cada carro terá uma cor aleatória e deixará uma linha desta cor que representa o seu caminho percorrido. A largura da linha

deixada representa a velocidade do carro nesse instante: quanto mais rápida, mais fina fica a linha. Para alem disso é representada uma seta que representa o vetor velocidade do carro no instante atual.

5.3 Menu Vista Por Carro

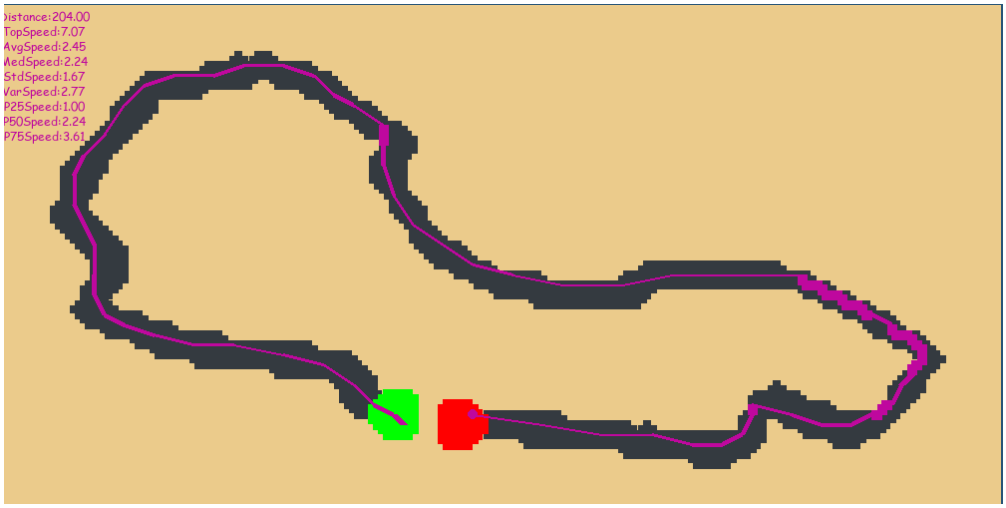

Figura 7: Menu Vista Por Carro

Neste menu é possível ver o caminho percorrido de cada um dos carros individualmente assim como algumas estatísticas sobre cada carro:

- Distancia percorrida
- Velocidade máxima
- Media da velocidade
- Mediana da velocidade
- Desvio padrão da velocidade
- Variância da velocidade
- Percentil 25 da velocidade
- Percentil 50 da velocidade
- Percentil 75 da velocidade

6 Manual de Utilização

6.1 Instalação

Para instalar todas as dependências do projeto, é necessário rodar o seguinte comando na raiz do repositório:

```
pip install -r requirements.txt
```

Para rodar o programa, executar

```
python main.py
```

O projeto foi testada para a versão 3.10 de Python.

6.2 Controlos

6.2.1 Controlos básicos

Todos os menus tem os mesmos controlos básicos:

- Q : Fechar o programa
- Esc : Voltar para o menu anterior
- R : Reiniciar a pagina atualizando informação
- Setas : Navegação dentro dos menus

6.2.2 Controlos Específicos

- Menu Principal
  - Enter : Fazer a simulação e ir para a vista de simulação
- Menu Simulação
  - P : Abrir o menu de vista por carro

7 Testes

7.1 Geração

Para isto foi feito um *script* num *jupyter notebook* que gera vários gráficos com a média para cada circuito e algoritmo das várias estatísticas.

7.2 Análise

Com os dados recolhidos conseguimos retirar algumas conclusões:

- Pesquisa em Profundidade
  - Se o circuito possuir soluções, uma é encontrada, embora nem sempre em tempo útil.
  - Possui o tempo computacional menos consistente, como pode ser observado no gráfico 28
  - A solução encontrada é por norma a pior em termos de custo dentre todos os algoritmos, como pode ser observado no gráfico 22. Isto deve-se a seguir um caminho sem olhar para o seu custo, apenas se chega a solução.
- Pesquisa em Largura
  - Se o circuito possuir soluções, uma é encontrada.
  - Em termos de custo computacional é o mais consistente dos algoritmos, sendo ligeiramente mais rápido que A* nos circuitos testados como pode ser visto em 28
  - Apresenta por norma o 2º menor custo, como pode ser observado no gráfico **??**. Isto deve-se às propriedades da pesquisa em largura: quando encontra uma solução esta será a mais curta em número de nodos percorridos.
- Gulosa
  - Apenas é encontrada uma solução se não houver paredes entre a partida e a chegada.
    - Como tem apenas em consideração a heurística e não o custo de avançar para o dado nodo, se existir uma parede entre os dois ficara num ciclo de ir contra a parede e voltar para a sua posição anterior.

  - Quando o algoritmo consegue encontrar uma solução, esta será mais rápida de computar, como se pode observar no gráfico 28. Isto deve-se a não existir a necessidade de processar tantos nodos.
  - A solução encontrada poderá não ser ótima, como pode ser observado no gráfico 22. Isto deve-se a ele conseguir viajar lateralmente contra uma parede, sendo penalizado pela colisão.

- A*
  - Se o circuito possuir soluções, uma é encontrada, embora nem sempre em tempo útil.
  - Este é o 2º melhor algoritmo em termos de custo computacional, superado apenas pela pesquisa em largura. Isto deve-se ao cálculo dos vários caminhos possíveis em simultâneo e a necessidade de substituir o custo do caminho até cada um dos nodos sempre que um melhor é encontrado.
    * Quanto mais próximo o início for do final sem um caminho direto, pior é em termos computacionais, como pode ser observado no gráfico 28 para o caminho badImage 9. Isto deve-e a compensar até certo ponto tentar ir contra a parede devido ao custo ser inferior do que afastar do ponto inicial.
  - Dentre todos os algoritmos, este apresenta o caminho com menor custo, como pode ser observado em 22.

- Pesquisa em Profundidade Iterativa
  - Se o circuito possuir soluções, uma é encontrada, embora nem sempre em tempo útil.
  - Este é o algoritmo com o pior custo computacional, não acabando em tempo útil na maioria dos circuitos testados.
    * Quanto mais comprido, em número de nodos, for o caminho mais curto até à chegada, pior é em termos computacionais. Isto deve-se à natureza iterativa do algoritmo, que alcança uma profundidade crescente em cada iteração.
  - Este algoritmo apresenta um caminho com custo comparável ao algoritmo de pesquisa em largura, como se pode ver no gráfico 22.

É preciso ter em conta que estes resultados referem-se a algoritmos em grafos já gerados e guardados sob o formato de listas de adjacência. Sob outras circunstâncias, que alterassem o custo base de visita ou travessia de um nodo, seriam observados resultados diferentes.

8 Conclusão

Com esta primeira fase do trabalho prático, as diferenças entre os diversos algoritmos de pesquisa explorados na Unidade Curricular.

O grupo está bastante satisfeito com o trabalho desenvolvido. Foram implementadas vários algoritmos, e os seus resultados e desempenho analisados. Além disso, as diversas funcionalidades adicionais tornam esta fase um sucesso, e uma excelente base para a próxima etapa, em que serão adicionados mais carros à simulação.

9 Anexos

9.1 Circuitos

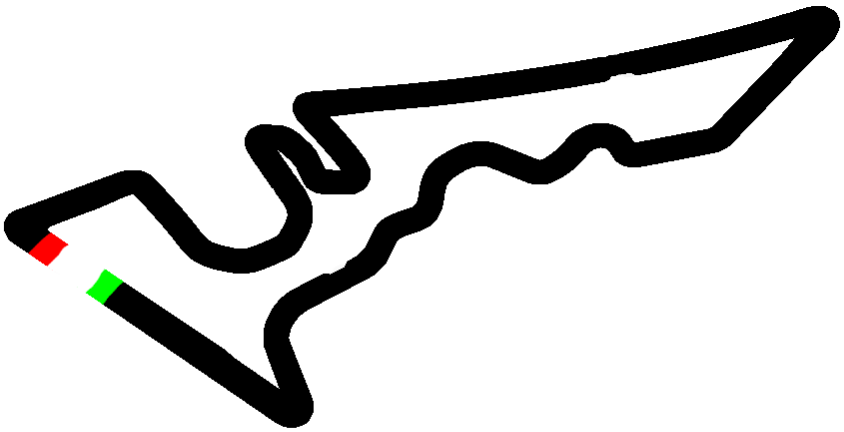

Figura 8: Circuito austin

Figura 9: Circuito badImage

Figura 10: Circuito bahrain

Figura 11: Circuito barcelona

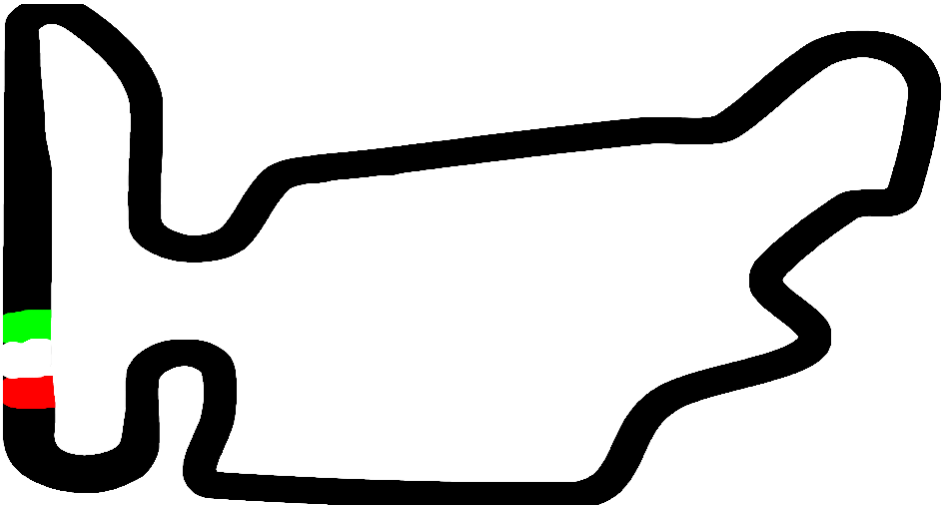

Figura 12: Circuito budapest

Figura 13: Circuito drag

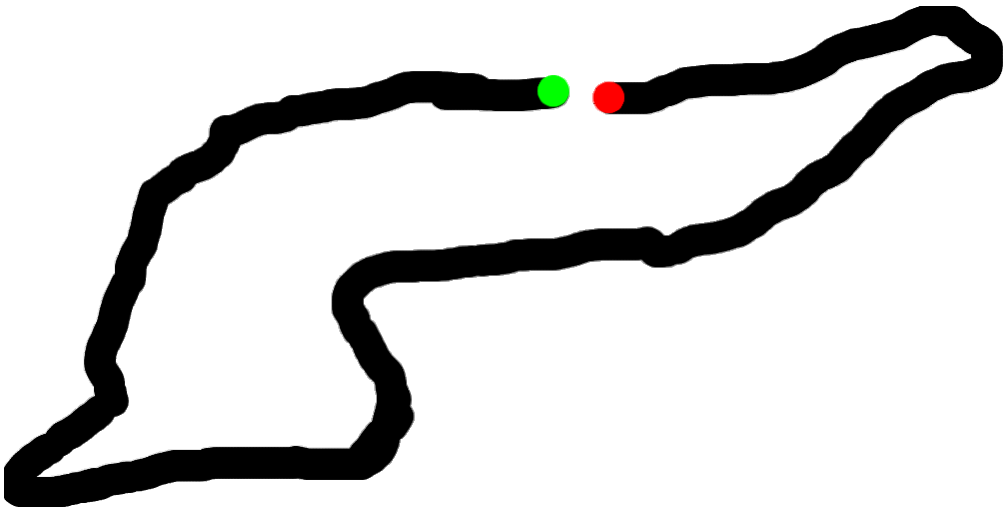

Figura 14: Circuito imola

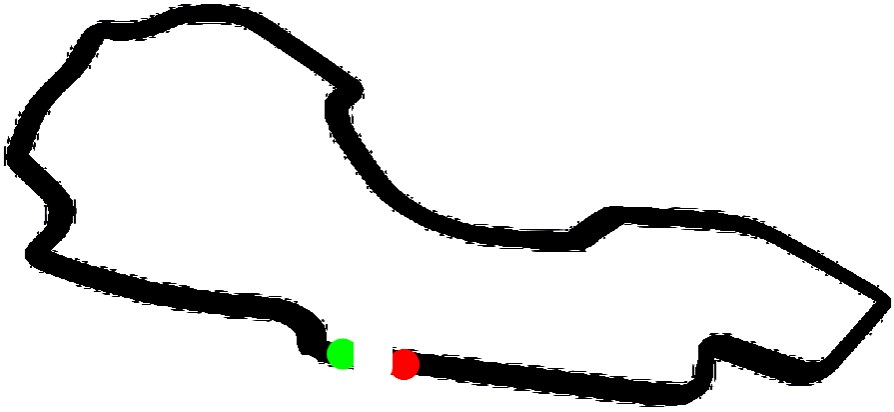

Figura 15: Circuito melbourne

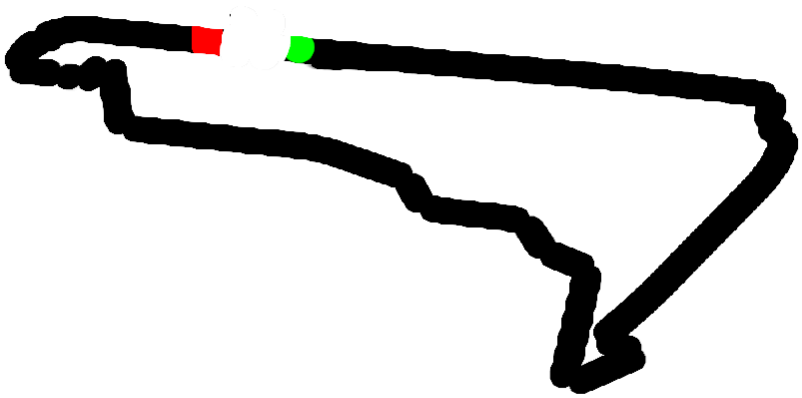

Figura 16: Circuito mexico

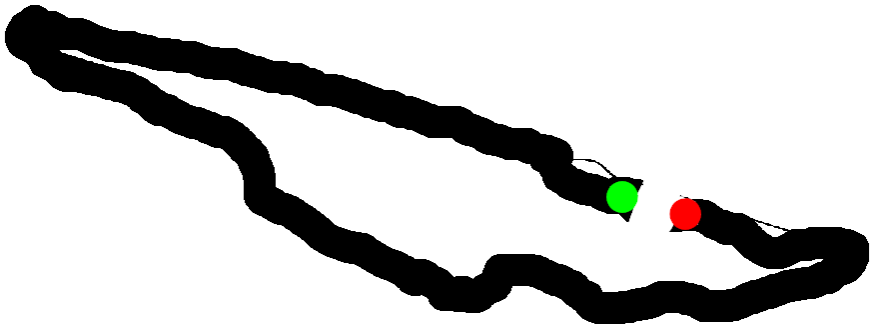

Figura 17: Circuito montreal

Figura 18: Circuito overkill

Figura 19: Circuito silverstone

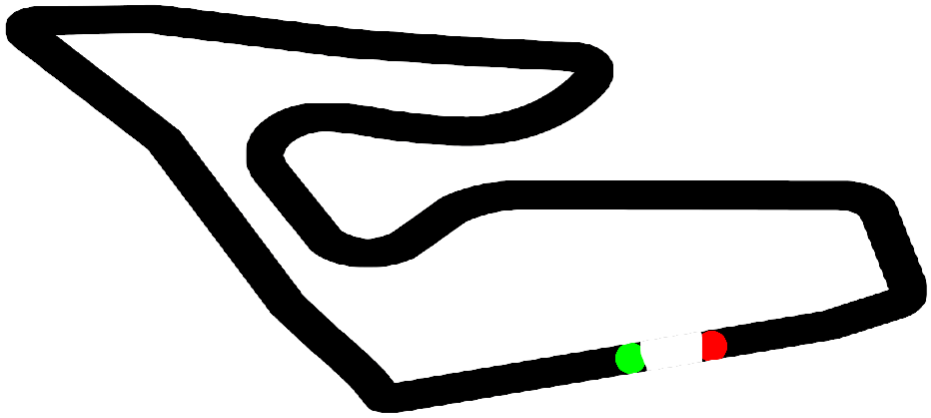

Figura 20: Circuito spielberg

9.2 Gráficos

Os seguintes dados foram gerados num computador com as especificações referidas na tabela 3

Sistema Operativo	ArcoLinux rolling [x86_64]
Kernel	5.15.79-1-lts
CPU	Intel Core i7-8750H
GPU	Intel UHD Graphics 630
RAM	2 * DDR4 Samsung M471A5244CB0-CTD 4 GB
SSD	WD Black SN750

Tabela 3: Especificações

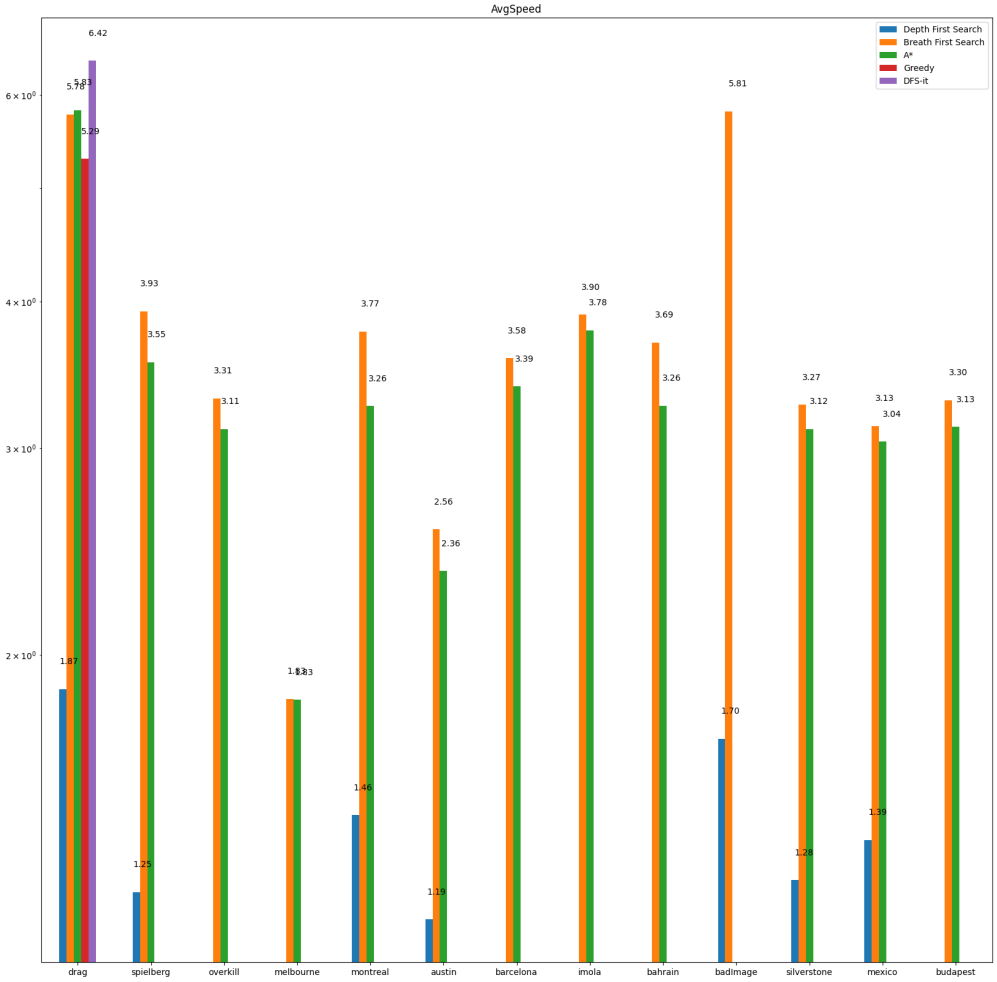

Figura 21: Estatística Velocidade média

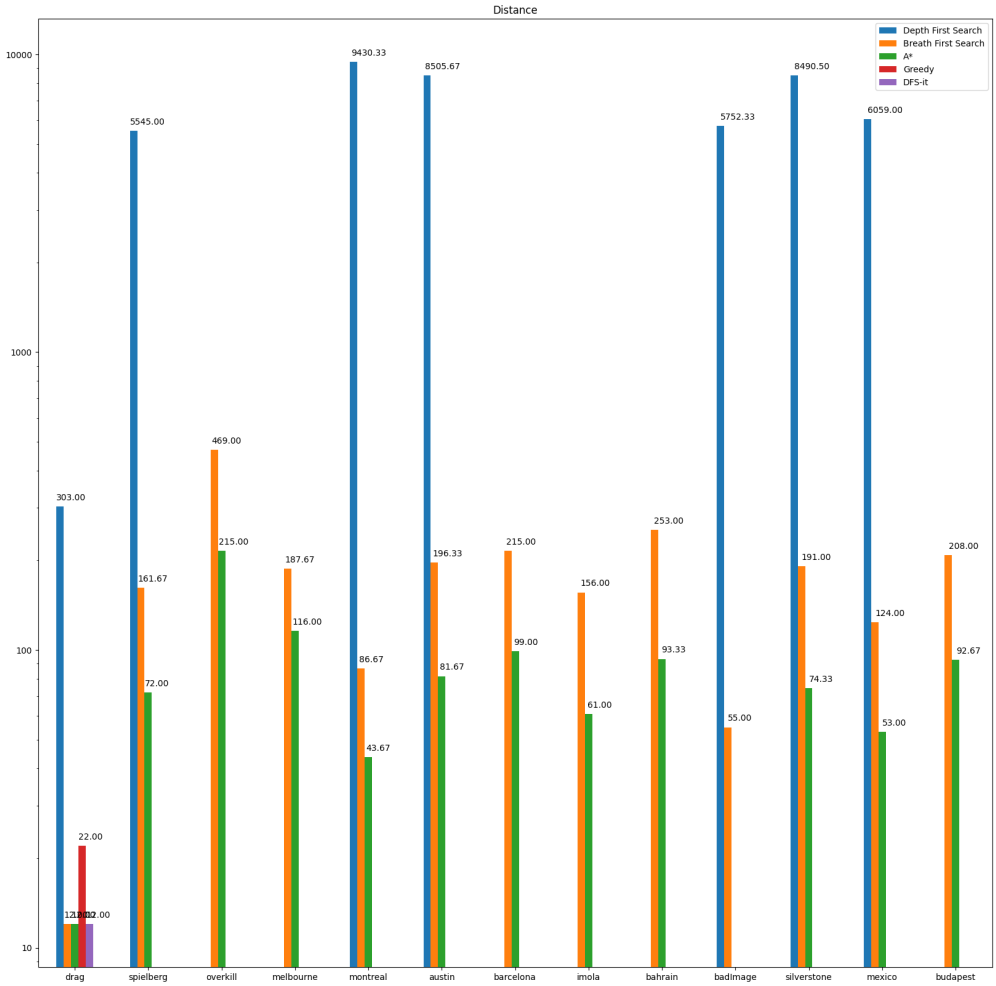

Figura 22: Estatística Distância

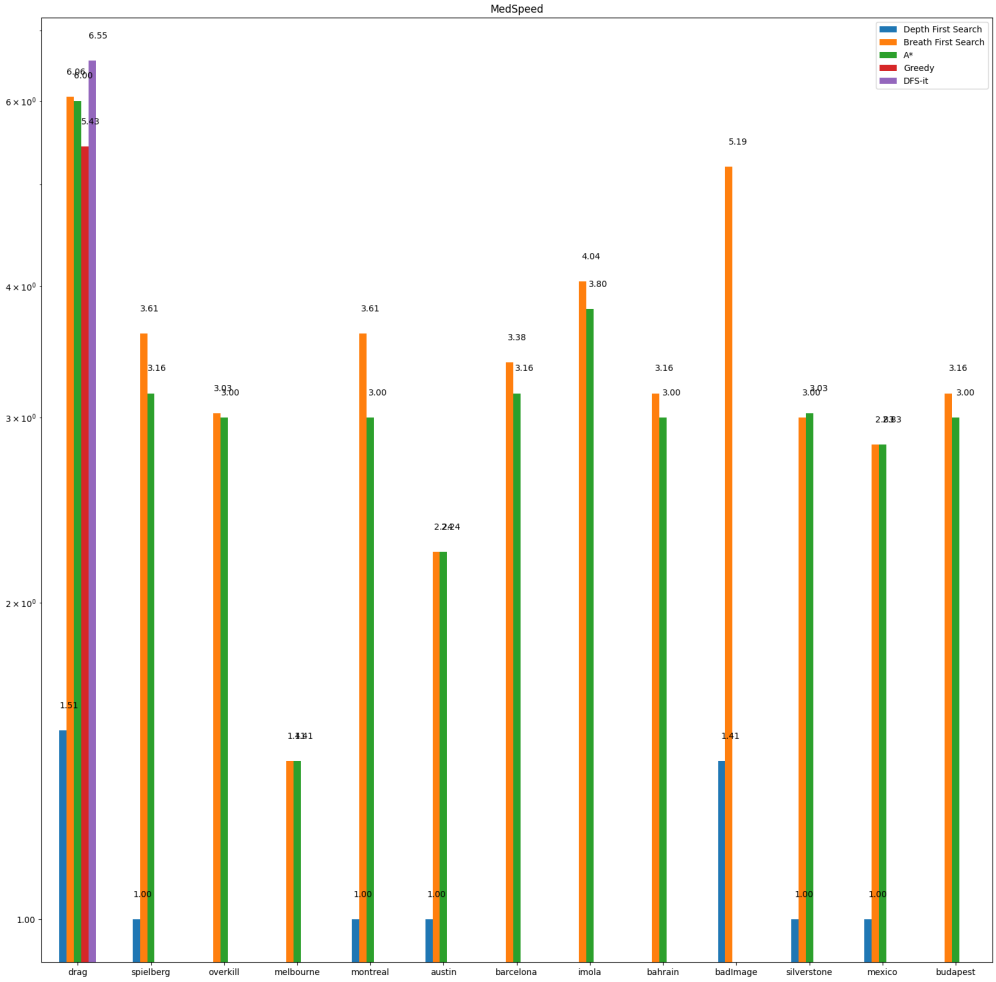

Figura 23: Estatística Velocidade mediana

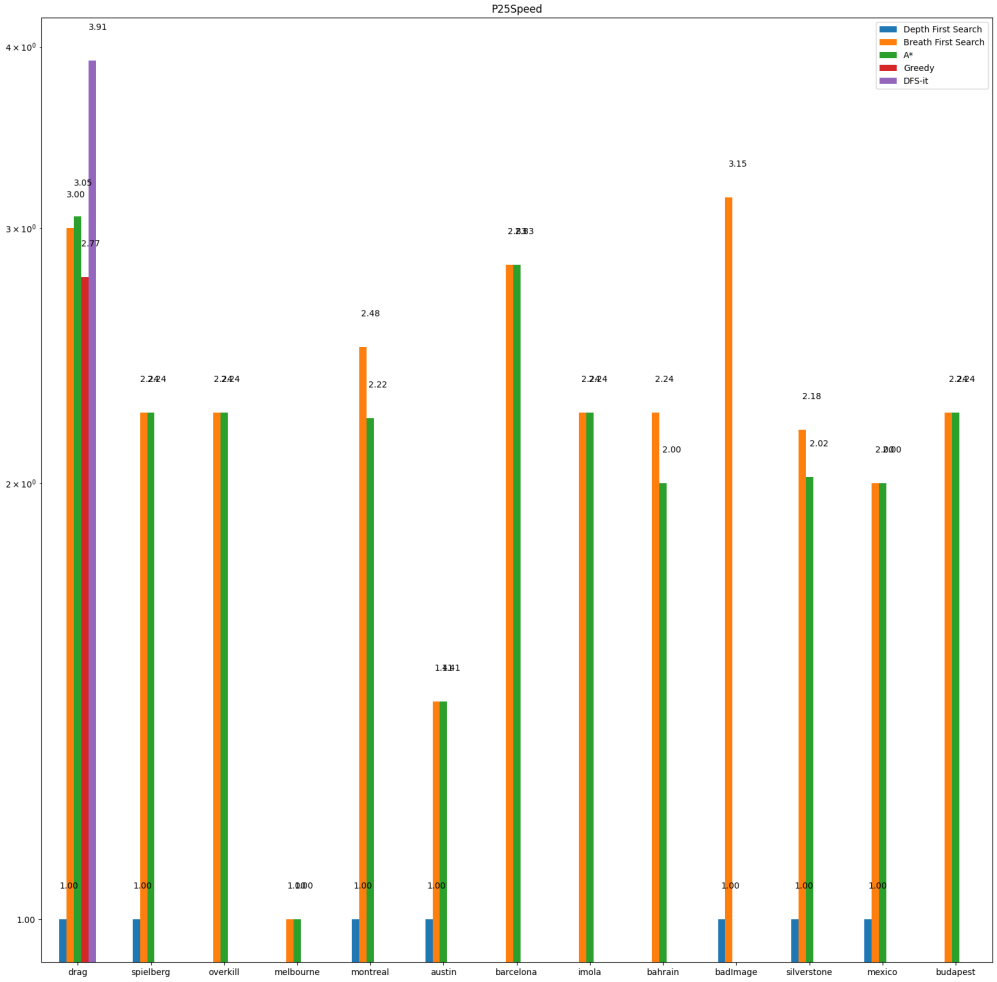

Figura 24: Estatística Per centil 25 da Velocidade

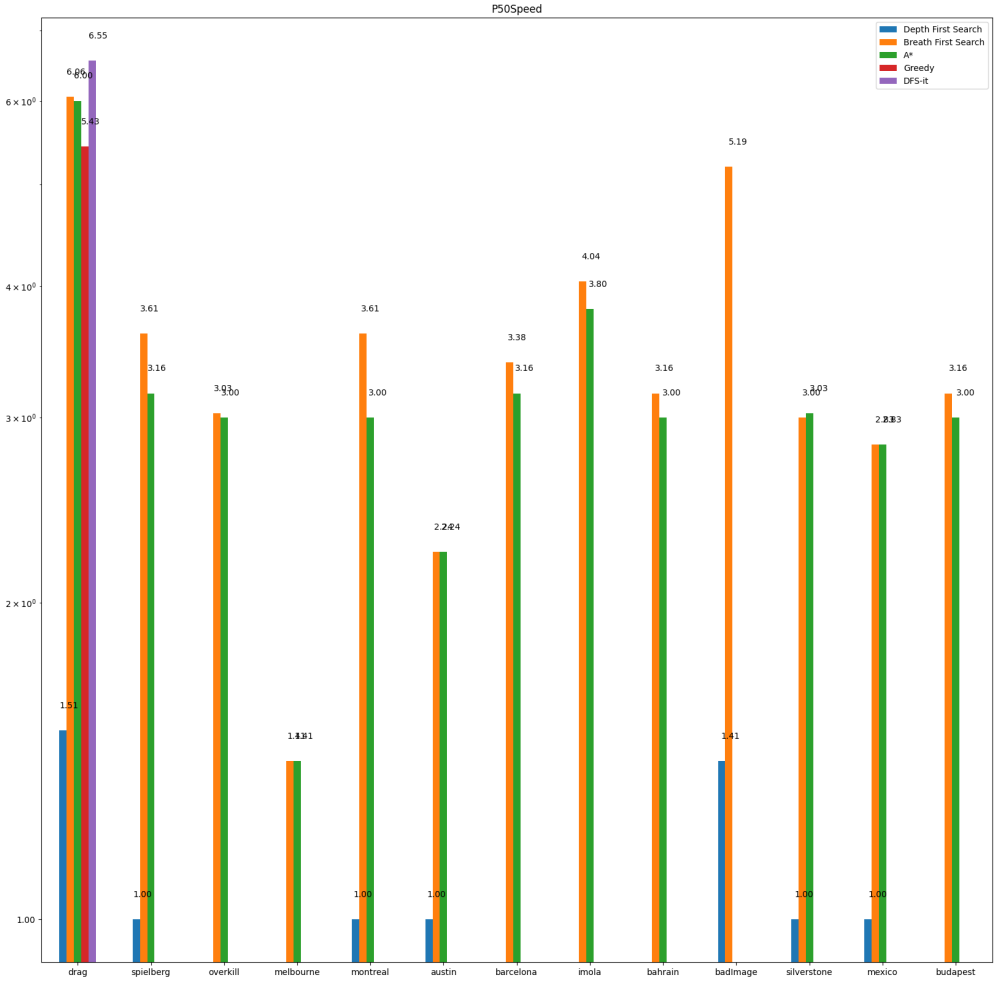

Figura 25: Estatística Per centil 50 da Velocidade

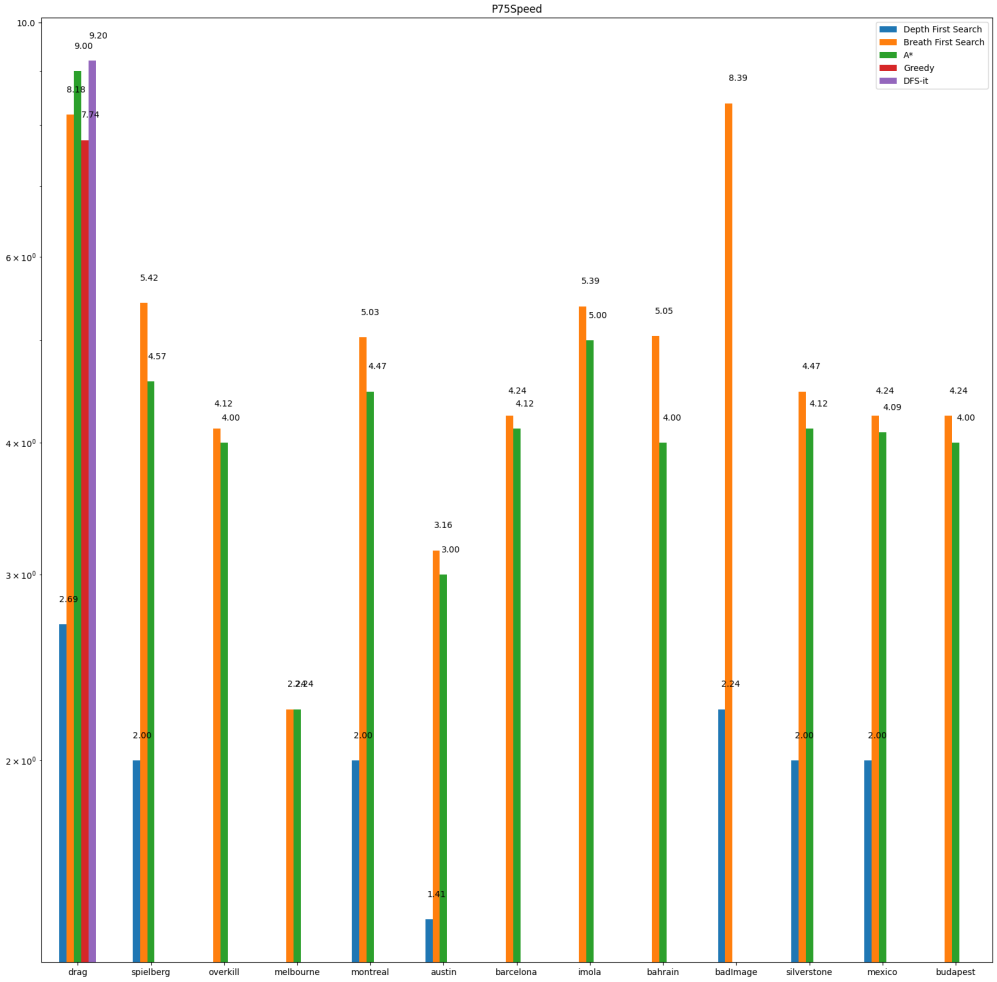

Figura 26: Estatística Per centil 75 da Velocidade

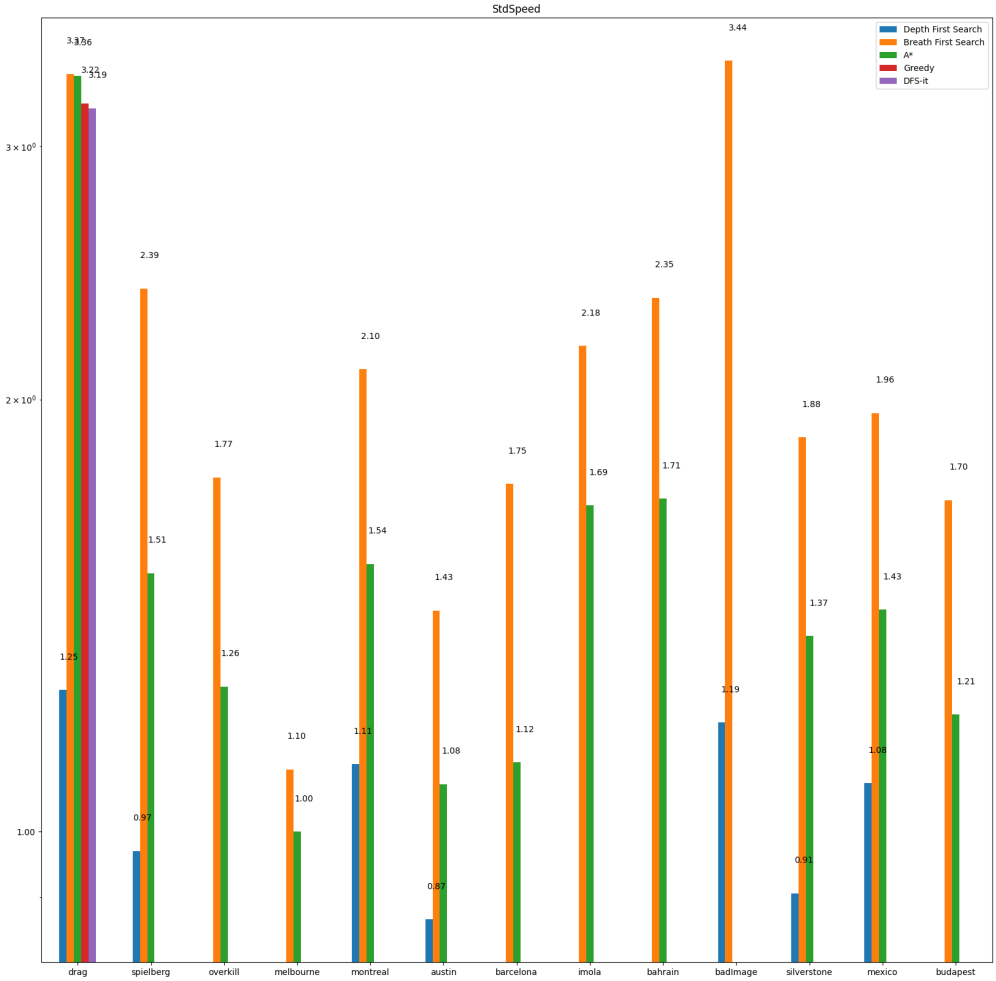

Figura 27: Estatística Desvio padrão da velocidade

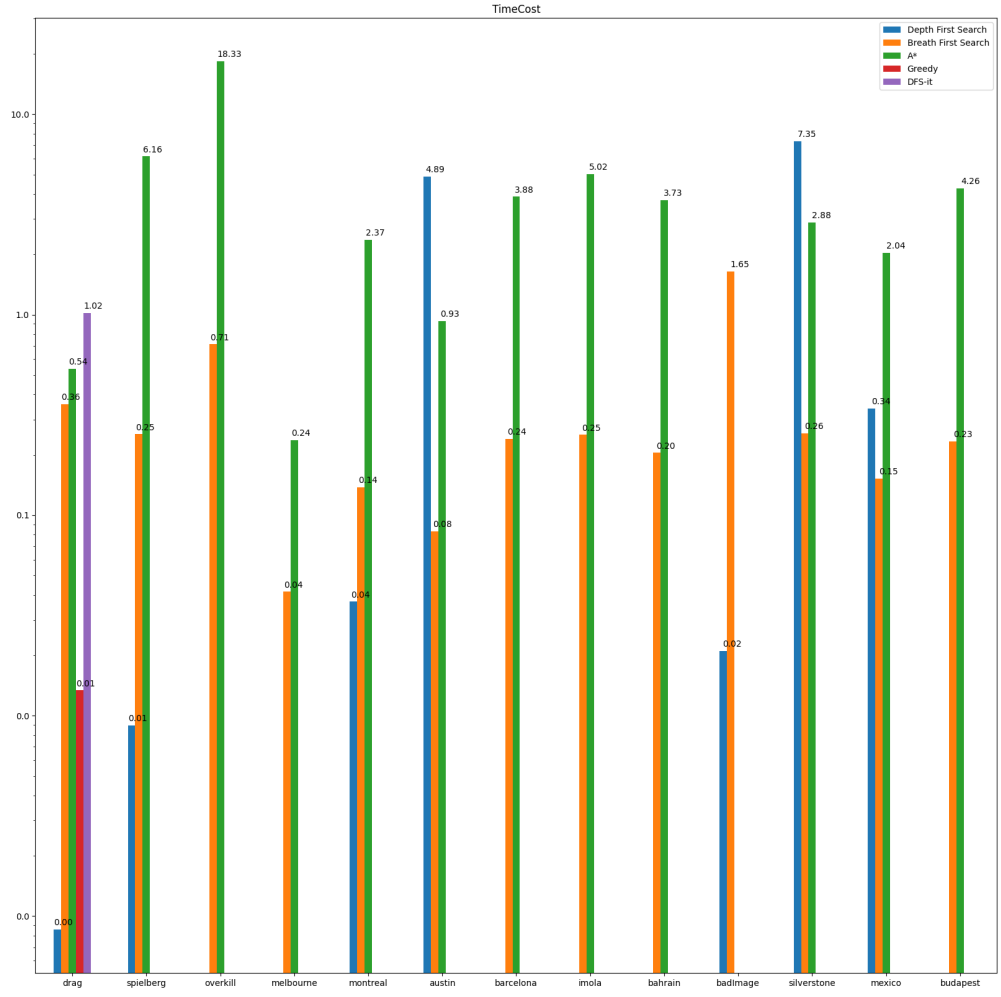

Figura 28: Estatística Tempo computacional

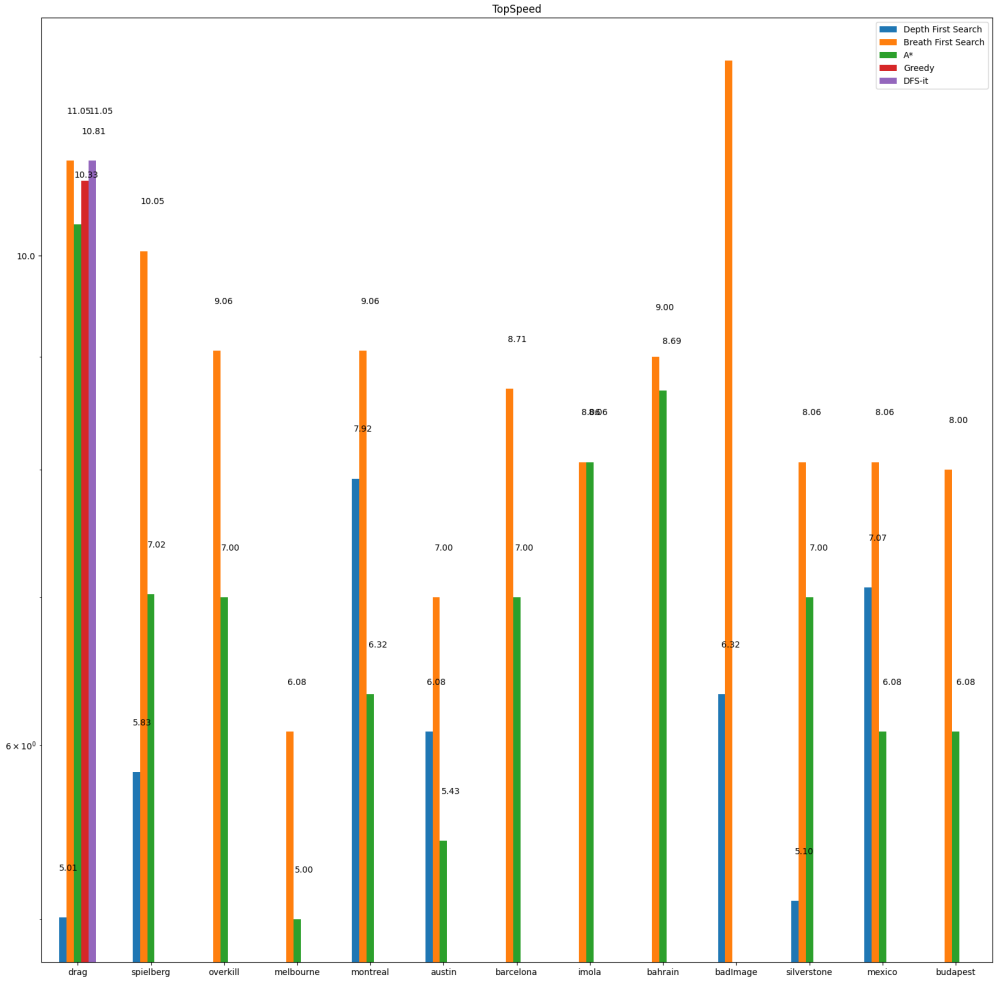

Figura 29: Estatística Velocidade máxima

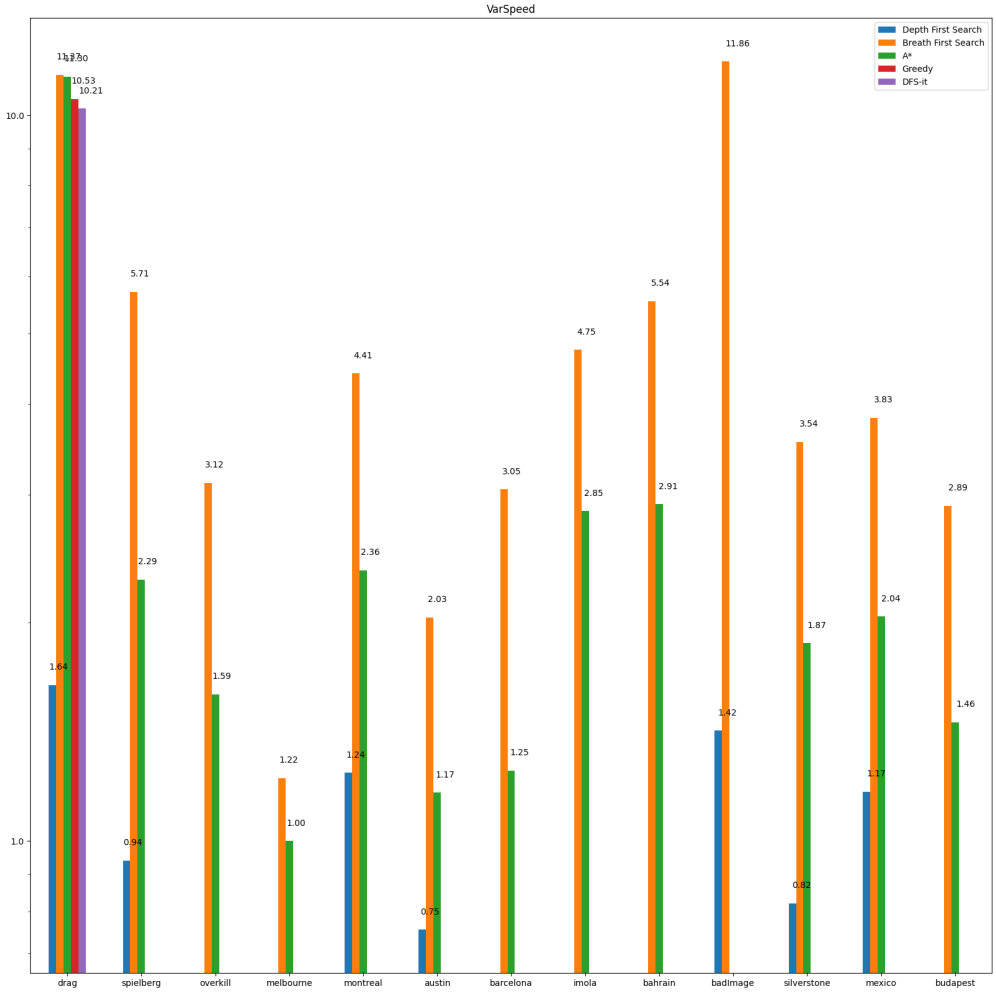

Figura 30: Estatística Variância da Velocidade