Uma imagem com texto

Descrição gerada automaticamente

Evolutionary Algorithms – a2rl

André Filipe Frade Guerra

Outubro/2022

Conteúdo

[Introdução 3](#_Toc117354189)

[Classes e métodos 3](#_Toc117354190)

[Exercício 1 5](#_Toc117354191)

[Resultados exercício 1 6](#_Toc117354192)

[Exercício 2 7](#_Toc117354193)

[Resultados exercício 2 8](#_Toc117354194)

[Exercício 3 9](#_Toc117354195)

[Resultados exercício 3 11](#_Toc117354196)

[Exercício 4 12](#_Toc117354197)

[Resultados exercício 4 14](#_Toc117354198)

[Conclusão 16](#_Toc117354199)

## Introdução

Neste relatório irá ser explicado resumidamente as classes criadas, funcionalidade dos métodos criados e por fim a execução e discussão dos resultados dos exercícios.

Este trabalho tem como objetivo implementar vários algoritmos que descobrem um padrão pré-definido/solução, e comparar o tempo e tentativas que cada um levou a processar até chegar à solução. Este padrão pode ter um tamanho variável e em cada posição terá uma cor que foi atribuída aleatoriamente.

## Classes e métodos

Para desenvolver os exercícios da segunda prática da disciplina de Introdução à Aprendizagem Automática foi desenvolvido um código em python usando o vsCode para realizar a mesma.



As seguintes classes foram criadas, sendo as 4 últimas auxiliares para as anteriores, que são os exercícios resolvidos.

Começando pela classe “Funcs.py”, contem métodos que são comuns nos exercícios:

Uma imagem com texto

Descrição gerada automaticamenteEste método recebe um valor numérico que vai construir um array com o tamanho das patterns a serem testadas.

Ex: se receber valor 9, irá retornar um array = [1, 2, 4, 6, 8, …, 18], basta duplicar o valor que foi recebido para percebermos qual o tamanho da última pattern.

Uma imagem com texto

Descrição gerada automaticamenteNeste método é retornada uma cor aleatória tendo em conta as cores possíveis que o são definidas no início do algoritmo, estas podem ir de 2 a 8 no máximo.

Uma imagem com texto

Descrição gerada automaticamenteEste método retorna uma pattern aleatória tendo em conta o tamanho da pattern em questão “cols” e quantas cores foram definidas no início do algoritmo.

Uma imagem com texto

Descrição gerada automaticamenteEste método retorna uma pattern com o tamanho definido, mas sem cores definidas.

Uma imagem com texto

Descrição gerada automaticamente

Este método retorna o valor de uma pattern tendo em conta a solução. Se a pattern for de tamanho 15, esse será o valor mais alto que será retornado pois significa que é igual à solução.

Uma imagem com texto

Descrição gerada automaticamenteEste método é o inverso do anterior, porque retorna 0 quando a pattern é igual a solução.

Uma imagem com texto, ecrã, laranja, captura de ecrã

Descrição gerada automaticamenteEste método recebe uma pattern, na qual será alterado numa das posições, aleatoriamente escolhida, por uma cor, também aleatoriamente escolhida.

Uma imagem com texto

Descrição gerada automaticamente

Este método recebe um array com X patterns (tendo em conta o exercício 3 serão 30 patterns). Destas 30 patterns são escolhidas 2 aleatoriamente. Na primeira pattern é escolhida uma posição aleatória que irá receber uma cor da segunda pattern que também foi escolhida aleatoriamente.

## Exercício 1

Presente no ficheiro exer1.py, este ficheiro começa por declarar vários valores iniciais que definem o algoritmo, como quantas patterns vão ser testadas, runs por pattern, quantas cores, e o tempo máximo que o algoritmo pode correr (neste caso 3600 segundos o que corresponde a 1 hora).

Uma imagem com texto

Descrição gerada automaticamente

De seguida, encontra-se o algoritmo implementado, que vai testar todas as patterns tendo em conta as runs que essa pattern deve ser testada. Para cada run é criada uma solução aleatória, que vai tentar ser descoberta criando-se uma pattern aleatória (linha 25). Na linha 28 e 29, encontra-se comentado os métodos que calculam a distância incremental e inversa, entre a solução e a pattern gerada. Esta pattern gerada aleatoriamente é sempre gerada até ser igual à solução definida.

Uma imagem com texto

Descrição gerada automaticamente

No fim serão mostrados 2 gráficos com boxplots, que contêm as tentativas e os tempos necessários para descobrir cada pattern a ser testada.

Uma imagem com texto

Descrição gerada automaticamente

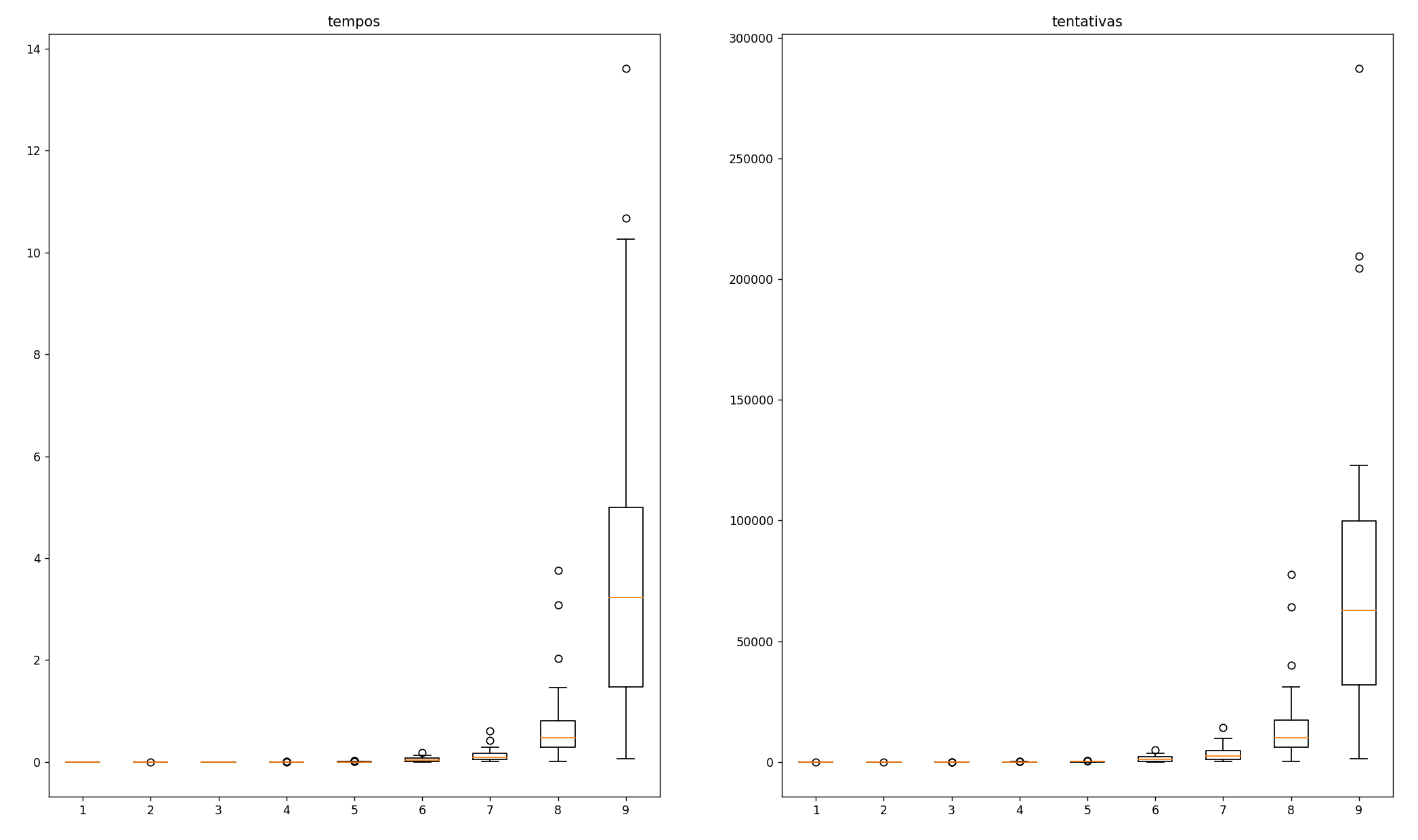
### Resultados exercício 1

O exercício 1 foi inicializado para testar até à pattern de tamanho 16.

Uma imagem com texto, interior

Descrição gerada automaticamente

Nos gráficos apresentados, podemos observar os resultados das tentativas e os tempos que cada tamanho das patterns demoraram a chegar à solução. Tendo a última solução um tempo médio de aproximadamente 3 segundos e 70 000 tentativas para chegar a solução.



## Exercício 2

Uma imagem com texto

Descrição gerada automaticamentePresente no ficheiro exer2.py, este ficheiro começa por declarar vários valores iniciais que definem o algoritmo, como quantas patterns vão ser testadas, runs por pattern, quantas cores, e o tempo máximo que o algoritmo pode correr (neste caso 3600 segundos o que corresponde a 1 hora). E adicionalmente, comparado com o exercício 1, foi definido também quantas vezes o algoritmo pode alterar uma posição aleatória de uma pattern até que esta esteja mais próxima da solução “runsForFlipOneBit”, se não mesmo encontrar a solução.

Após a declaração e definição das variáveis, encontra-se o algoritmo que irá tentar descobrir a solução, alterando apenas uma posição de uma pattern regada aleatoriamente, que para quando o limite de tentativas de alterar uma posição foi alcançado “runsForFlipOneBit”, ou a solução foi encontrada. Quando a alteração de uma posição fica mais próxima da solução, esta é armazenada e utilizada na próxima iteração de alterar uma outra posição.

Uma imagem com texto

Descrição gerada automaticamente

No fim serão mostrados 2 gráficos com boxplots, que contêm as tentativas e os tempos necessários para descobrir cada pattern a ser testada.

Uma imagem com texto

Descrição gerada automaticamente

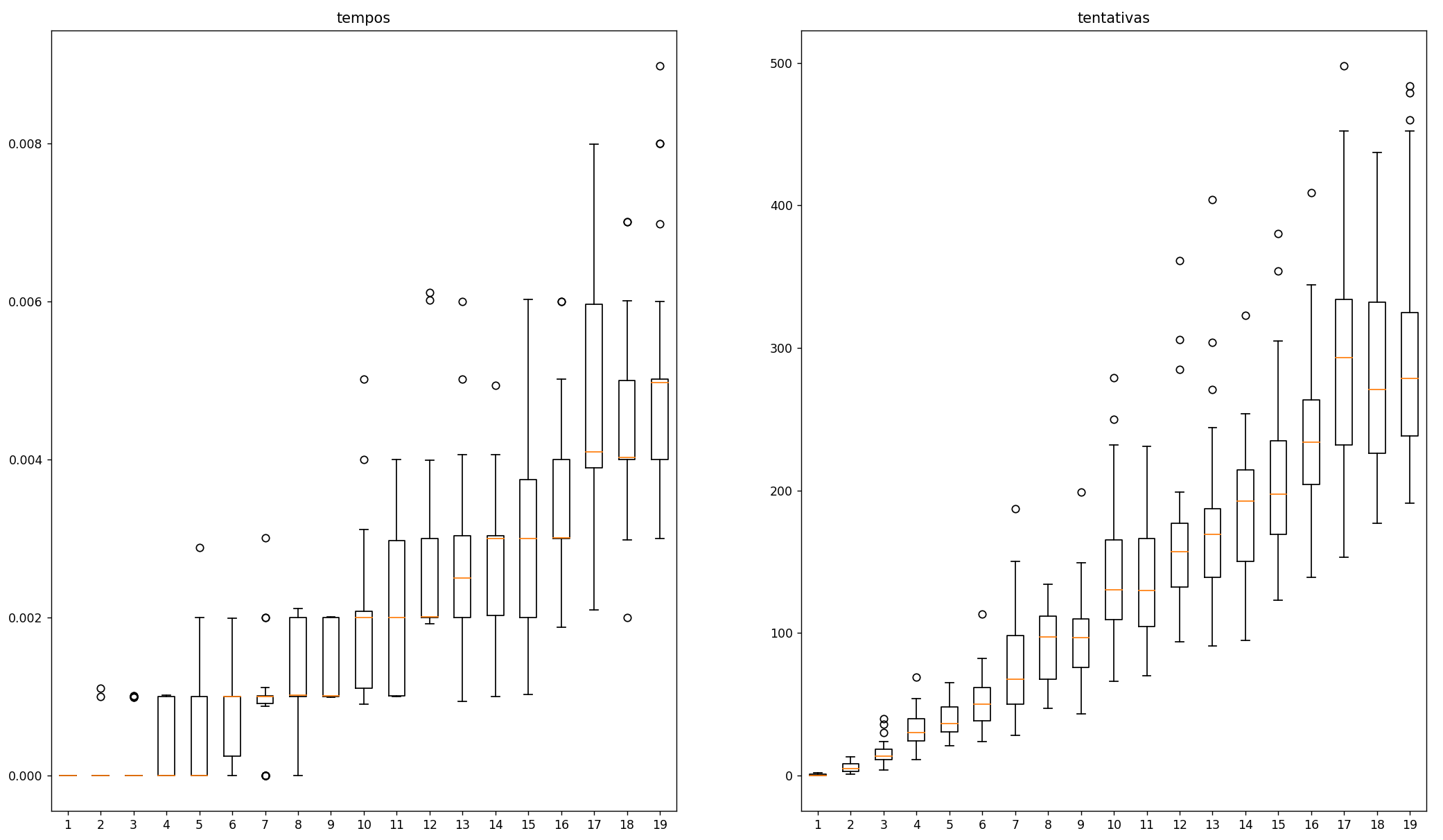
### Resultados exercício 2

Tendo em conta a velocidade com que o algoritmo chega a solução o tamanho das patterns a ser testado foi aumentado para 18 (1, 2, 4, 6, …, 36), sendo o 36 a pattern maior a ser testada.

Uma imagem com texto

Descrição gerada automaticamente

Com os gráficos obtidos, podemos observar o tempo e as tentativas a aumentarem quando a pattern aumenta de tamanho, este algoritmo chega à solução com a pattern mais alta (pattern com 36 posições), num tempo médio de 0.005 segundos e 280 tentativas. Comparando como exercício 1, este é significativamente mais rápido, pois quando se altera uma das posições da pattern, esta só é guardada caso esteja mais próxima da solução.



## Exercício 3

Presente no ficheiro exer3.py, este ficheiro começa por declarar vários valores iniciais que definem o algoritmo, como quantas patterns vão ser testadas, runs por pattern, quantas cores, e o tempo máximo que o algoritmo pode correr (neste caso 3600 segundos o que corresponde a 1 hora). E adicionalmente, comparado com o exercício 1, foi definido também o tamanho da população a ser gerado por cada pattern a ser testada “howManyPatternsPopulation”.

Uma imagem com texto

Descrição gerada automaticamente

De seguida, encontra-se o algoritmo que tenta encontrar a solução, organizando a população das patterns geradas aleatoriamente pelo valor mais próximo da solução. A cada tentativa de descobrir a solução, o algoritmo escolhe as 30 melhores patterns, que de seguida são usadas para gerar 70 novas patterns alterando uma das posições aleatoriamente. Este processo volta a repetir-se selecionando novamente as melhores 30 patterns até que a solução seja encontrada. A solução é declarada quando a primeira posição da população de patterns é igual a solução, pois essa população de patterns está organizada da mais próxima até a mais longe da solução.

Uma imagem com texto

Descrição gerada automaticamente

No fim serão mostrados 2 gráficos com boxplots, que contêm as tentativas e os tempos necessários para descobrir cada pattern a ser testada.

Uma imagem com texto

Descrição gerada automaticamente

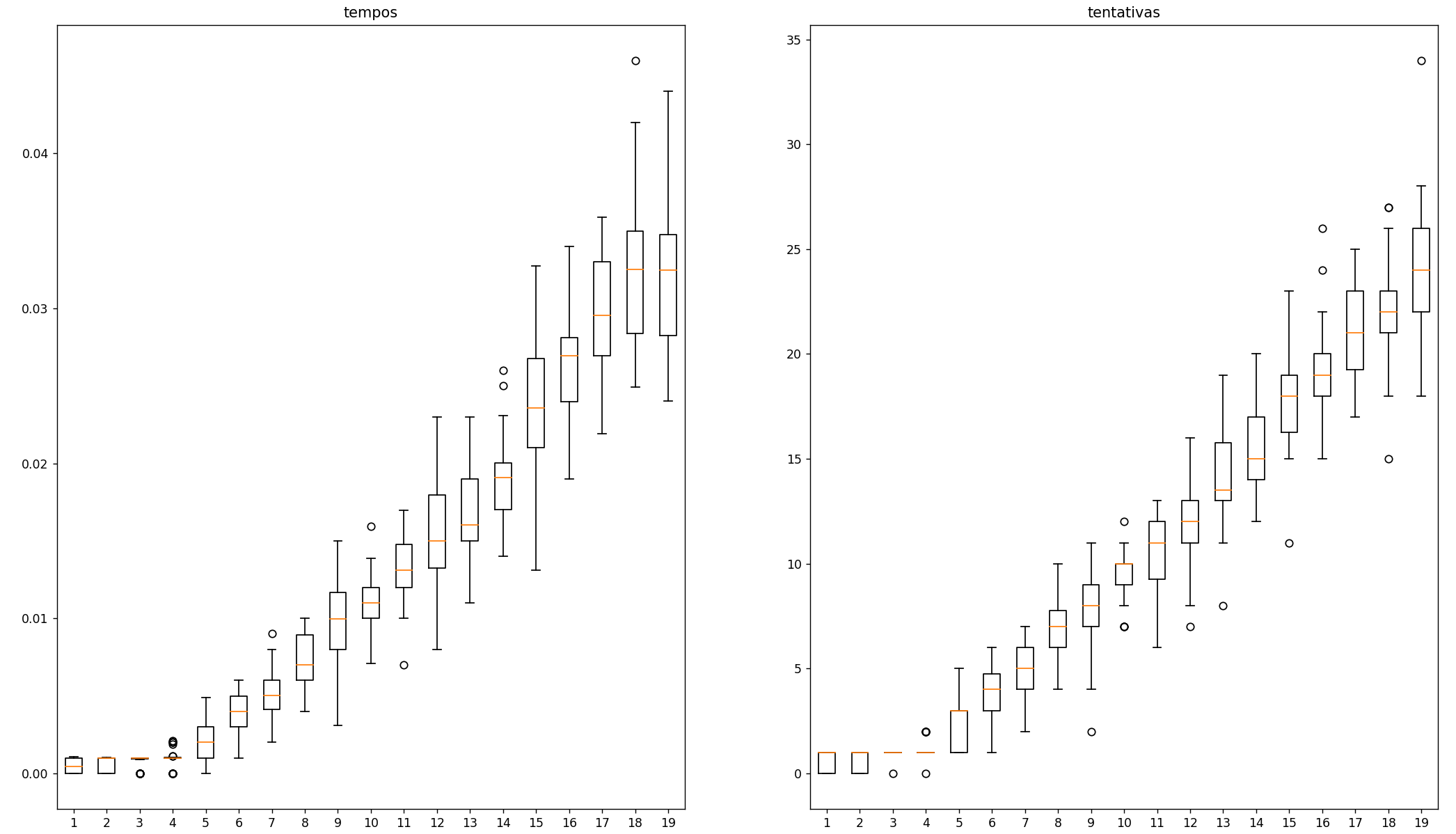
### Resultados exercício 3

No exercício 3, as variáveis de inicialização, permaneceram iguais às do exercício 2, mais a variável que define o tamanho da população a ser gerada, neste caso 100.

Uma imagem com texto

Descrição gerada automaticamente

Com os gráficos seguintes, podemos concluir que apesar do tempo ter aumentado, as tentativas reduziram significativamente, o que no fim originou um algoritmo ainda mais rápido, comparado com os exercícios anteriores. Esta diminuição das tentativas, pode explicar-se, devido a este algoritmo guardar recursivamente as 30 melhores patterns em 100, e criar uma população fazendo a alteração de uma posição aleatória nessas 30 patterns, que também são escolhidas aleatoriamente.



## Exercício 4

Presente no ficheiro exer4.py, este ficheiro começa por declarar vários valores iniciais que definem o algoritmo, como quantas patterns vão ser testadas, runs por pattern, quantas cores, e o tempo máximo que o algoritmo pode correr (neste caso 3600 segundos o que corresponde a 1 hora). Adicionalmente, comparado com o exercício 3, foram adicionadas 2 variáveis que definem a percentagem das melhores patterns presentes na população que irão gerar a próxima população, como também a percentagem das novas petterns a serem criadas, neste caso está definido escolher as melhores 30% da população, e criar 70% patterns (estas duas variáveis devem somar no total 1).Uma imagem com texto

Descrição gerada automaticamente

De seguida, encontra-se o algoritmo que tenta encontrar a solução, e semelhante ao exercício 3, organiza a população das patterns geradas aleatoriamente pelo valor mais próximo da solução. A cada tentativa de descobrir a solução, o algoritmo escolhe as 30 melhores patterns, que de seguida são usadas para gerar 70 novas patterns.

Estas 70 novas patterns são geradas a partir da escolha aleatória de duas patterns das 30 melhores, substituído uma posição aleatória da primeira pela posição aleatória de segunda (ver descrição do método “CrossOver”).

Este processo volta a repetir-se selecionando novamente as melhores 30 patterns até que a solução seja encontrada. A solução é declarada quando a primeira posição da população de patterns é igual a solução, pois essa população de patterns está organizada da mais próxima até a mais longe da solução.

Uma imagem com texto

Descrição gerada automaticamente

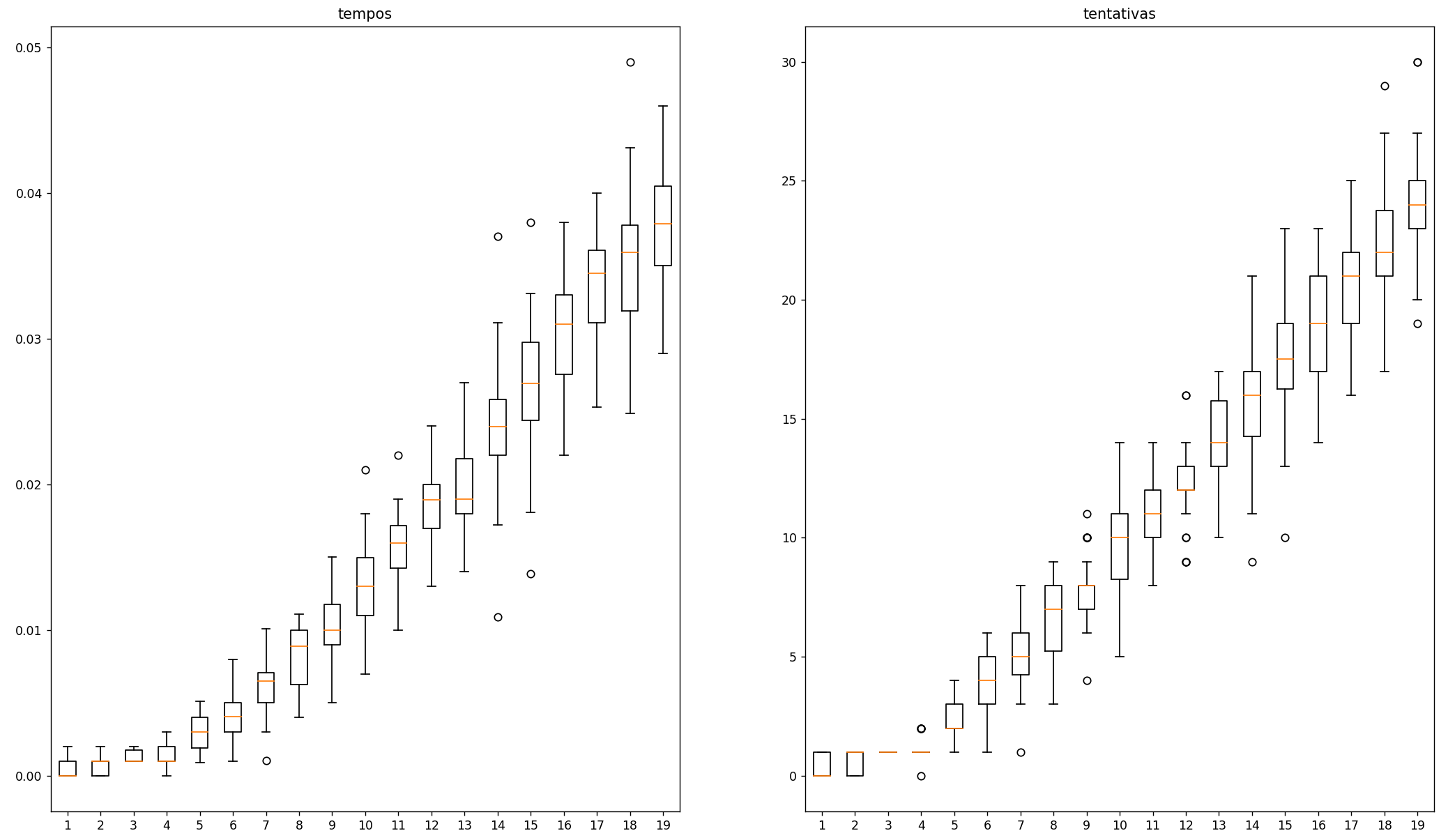
### Resultados exercício 4

No exercício 4, utilizando as variáveis de inicialização, permaneceram iguais ao exercício 3, com a adição de duas novas, a percentagem da população que deve ser guardada para criar a próxima geração (30% = 0.3), e a percentagem a ser criada (70% = 0.7).

Uma imagem com texto

Descrição gerada automaticamente

Nos gráficos seguintes, podemos observar valores muito semelhantes ao exercício 3, mesmo o algoritmo sendo um pouco diferente do anterior. A semelhança entre o exercício anterior é a percentagem a ser usada para guardar e criar a população, 30% e 70% respetivamente, apesar de neste, as posições das patterns serem alteradas tendo em conta outra pattern também incluída nas 30% melhores.

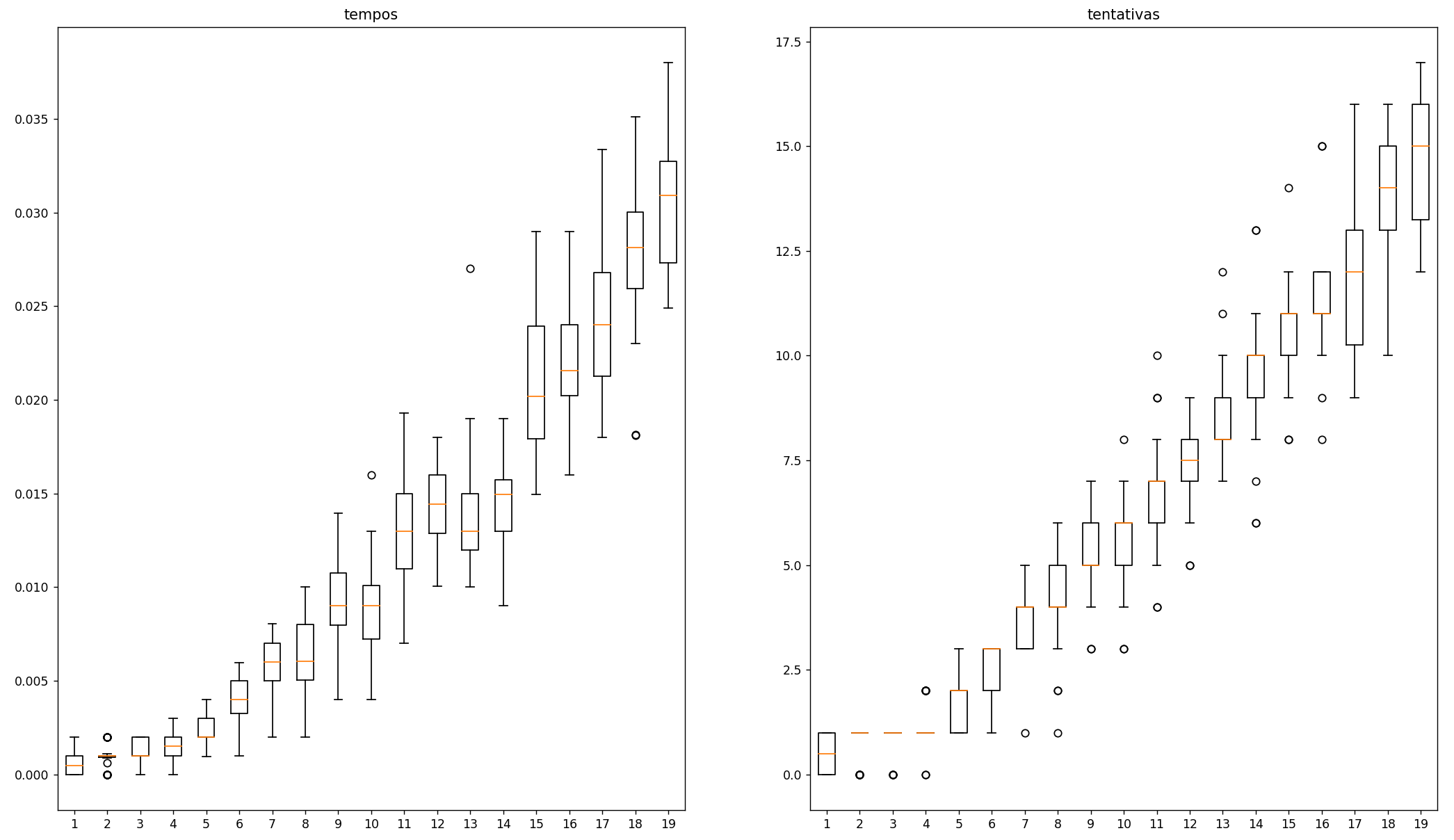


Para compararmos a percentagem de patterns guardadas e criadas, as variáveis de inicialização foram alteradas, de maneira que as patterns guardadas correspondem a 10%, e as criadas a 90%.

Uma imagem com texto

Descrição gerada automaticamente

Depois da execução do algoritmo, podemos observar que guardando apenas as 10% mais próximas da solução, este converge para a solução em menos tempo e tentativas.



## Conclusão

Inicialmente o código foi feito para poder trabalhar com várias cores, assim sendo possível em cada posição da pattern poderem existir até 8 cores, os exercícios anteriores foram executados sempre apenas com 2 cores possíveis.

Para podermos observar mais facilmente a relação entre a quantidade de cores e a velocidade, foi criado um rating que vai ser calculado da seguinte forma:

rating = (média dos tempos \* 500) + média das tentativas



Uma imagem com texto

Descrição gerada automaticamente

|  |  |  |  |
| --- | --- | --- | --- |
|  | 2 cores | 3 cores | 4 cores |
| Exercício 1 | 10421 | 10674 | --- |
| Exercício 2 | 134 | 201 | 266 |
| Exercício 3 | 16 | 27 | 38 |
| Exercício 4 30% | 16 | 28 | --- |
| Exercício 4 10% | 11 | 19 | --- |

Os dados não apresentados na tabela (---), são devido ao tempo elevado que o exercício 1 leva a chegar à solução com 4 cores, e devido a uma falha que pode estar a acontecer no exercício 4.

Esta falha pode ser explicada porque no exercício 4, a criação da população das patterns pode não conter uma das cores da solução, e também, como as novas patterns são geradas a partir das 10% ou 30% melhores, essas podem também não conter a cor, criando gerações sem a cor necessária para a solução. Ao contrário do exercício 3 em que a posição da pattern é alterada para uma cor aleatória das possíveis.

Assim, podemos concluir, que para um elevado número de cores o exercício 3 é mais indicado. Não apresenta falhas independentemente da quantidade de cores, como também, apresenta um rating inferior aos restantes exercícios 1 e 2.