Optimização na Programação de uma Conferência

Relatório Final

Inteligência Artificial

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Elementos do Grupo:

Anderson Rogério da Silva Gralha– 201710810– up201710810–

António Miguel Silva Pereira – 201307910– up201307910–

Vitor Emanuel Fernandes Magalhães – 201503447– up201503447–

20 de maio de 2018

Índice

1. Objetivo

Este projeto foi desenvolvido no âmbito da cadeira de Inteligência Artificial e tem como objetivo a aplicação de algoritmos de otimização na organização de uma conferência. Pretende-se, com isto, resolver a problemática da organização de uma conferência utilizando Algoritmos Genéticos, uma técnica que foi ensinada nas aulas.

2. Especificação

2.1. Análise do tema

Como mencionado anteriormente, este projeto consiste na otimização da programação de uma conferência com duração, geralmente de três dias, aplicando Algoritmos Genéticos no contexto do problema.

Em cada dia da convenção, podem existir até M sessões em paralelo delimitadas pelo número de salas disponíveis. As sessões contém um tema e duram no máximo 2 horas. Estas são compostas por papers, que podem ser short(com uma duração de 20 minutos) ou full(com uma duração de 30 minutos) e contêm, no mínimo 2 full-papers.

Durante o dia, pode haver sessões em quatro horários específicos: dois horários de manhã e dois horários de tarde. Estes horários são separados por coffee breaks e almoço(entre o último horário da manhã e o primeiro horário da tarde).

Cada paper é composto por um ou mais temas e vários autores, juntamente com a sua duração. Um paper também contém um apresentador, que é autor ou coautor desse paper.

2.2. Ilustração de cenários

Este programa pode ser aplicado na simulação de uma conferência, por exemplo na Feira Internacional de Lisboa, para facilitar o processo de organização da mesma.

Para isto, basta alterar os valores que são utilizados no algoritmo, como os temas disponíveis, o número de dias, o número de sessões no mesmo horário e o número de papers numa sessão.

Para definições mais avançadas, é possível alterar o tipo de seleção dos indivíduos, o número de iterações e o peso de cada componente no cálculo do melhor indivíduo.

2.3. Abordagem

Perante o problema, foi decidida a utilização de Algoritmos Genéticos. Estes algoritmos foram baseados no conceito e evolução e na seleção natural de Charles Darwin e desenvolvidos por John Henry Holland.

As seguintes instruções descrevem, muito sucintamente, o algoritmo:

1. Obtém-se a população inicial.
2. A solução é suficientemente boa?
3. Fase de Selecção.
4. Fase de Emparelhamento.
5. Fase de Crossing-Over.
6. Fase de Mutação.
7. A solução é suficientemente boa?
    (a) Se sim, termina.
    (b) Se não, volta a 3.

2.3.1. População Inicial

Nesta primeira fase, é necessário definir um indivíduo pertencente à população. Um indivíduo é tipicamente representado em forma binária, onde cada 0 e 1 é nomeado de alelo. Esta representação permite definir um indivíduo como um cromossoma.

Cada indivíduo é composto por elementos que o definem, nomeados genes, que são compostos por um ou mais alelos.

Um indivíduo representa uma solução possível do problema, composto por uma *string* composta pelos dias e pelas suas respectivas sessões da conferência.

Para a população inicial, decidiu-se que o seu tamanho fosse introduzido pelo utilizador. A *string* de cada indivíduo é gerada aleatoriamente e posteriormente analisada.

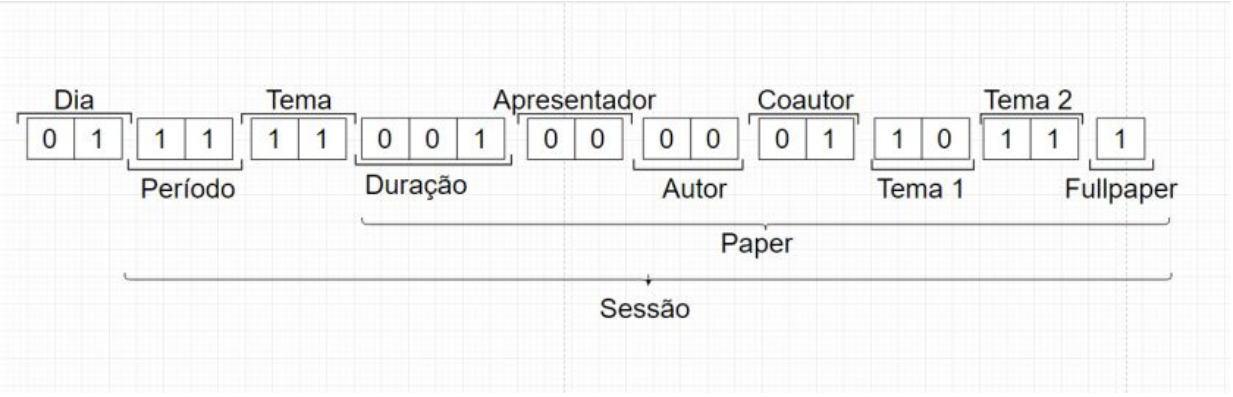

Imagem 1: Exemplo de Cromossoma com 1 dia, 1 sessão e 1 paper.

A tabela abaixo representa as possíveis combinações de papers, as suas durações e a sua representação em binário, com os dois full-papers obrigatórios subentendidos.

Combinação	Duração	Representação
20 + 20 + 20	60	111
30 + 30	60	110
20 + 30(ou 30 + 20)	50	101
20 + 20	40	100
30	30	011
20	20	010
0	0	001

Tabela 1: Tabela de combinações de Papers

2.3.2. Função de Avaliação

A função de avaliação é uma função que permite calcular e atribuir um valor(fitness score) a um indivíduo. Esta função depende do contexto do problema e a sua implementação varia de problema para problema.

Para a função de avaliação deste projeto, são aplicadas diversas verificações e filtros para garantir que os indivíduos selecionados para a próxima geração possuam características mínimas, ou seja, características que, caso não estejam presentes nos indivíduos, não podem ser replicadas de forma alguma nas próximas gerações. Para uma conferência, é essencial que:

- Haja, no mínimo, dois full-papers em cada sessão.
- O apresentador do paper, que pode ser coautor ou autor, não possa estar em duas apresentações ao mesmo tempo.
- Verificar que o apresentador do paper seja um autor ou coautor deste mesmo.
- Não pode existir o mesmo autor repetido dentro do mesmo paper.

Caso haja algum indivíduo no qual se verifica uma destas condições, é lhe atribuído um fitness score de 0. Os indivíduos que satisfaçam as condições acima recebem um fitness score relativo ao tema da sessão: é atribuído um fitness score maior a um indivíduo cujo tema de cada sessão esteja nos temas dos papers dessas sessões.

Na última etapa, é necessário averiguar as diferenças de tempo entre as apresentações dos papers durante todos os dias, gerando o valor máximo caso todos os dias contenham a mesma duração. No final soma-se o fitness score das diferenças de tempo com os temas das sessões e obtém-se o fitness score final do indivíduo.

2.3.3. Fase de Seleção

No início do programa, o utilizador pode escolher o tipo de seleção para o algoritmo. É na fase de seleção que a diferença é sentida.

Numa seleção probabilística, é criada uma "roleta", limitada pelas probabilidades de cada indivíduo. Estas probabilidades são representadas pela seguinte fórmula, onde $f(Ci)$ representa o fitness score do indivíduo e $\sum_{i=1}^{M} f(Ci)$ representa a soma de todos os fitness scores da população.

$$P(Ci) = f(Ci) / \sum_{i=1}^{M} f(Ci)$$

Numa seleção elitista, escolhe-se os N melhores cromossomas, ditado também por input do utilizador. Os restantes indivíduos são obtidos pela "roleta", tal como a seleção probabilística. No fim, os indivíduos obtidos na roleta juntam-se aos indivíduos selecionados como melhores.

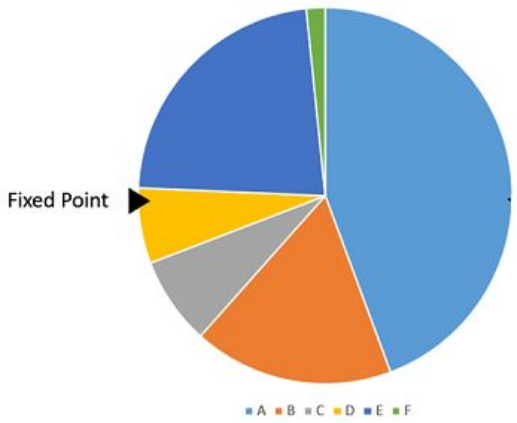

Imagem 2: Exemplo de uma selection wheel com diferentes probabilidades.

2.3.4. Fase de Emparelhamento

Nesta fase, há uma probabilidade de alguns indivíduos se juntarem para cruzamento. Caso haja um número ímpar de indivíduos para o cruzamento, descarta-se o último indivíduo.

2.3.5. Fase de Cruzamento

Na fase de cruzamento, existem várias estratégias possíveis para originar filhos de dois indivíduos. Neste projeto, foi utilizada a estratégia HUX(Half Uniform Crossover). Com uma probabilidade Px, o gene de um indivíduo é trocado pelo gene do outro e vice-versa.

No fim, resultam dois filhos que substituem os pais na população, de modo a manter o tamanho original da população.

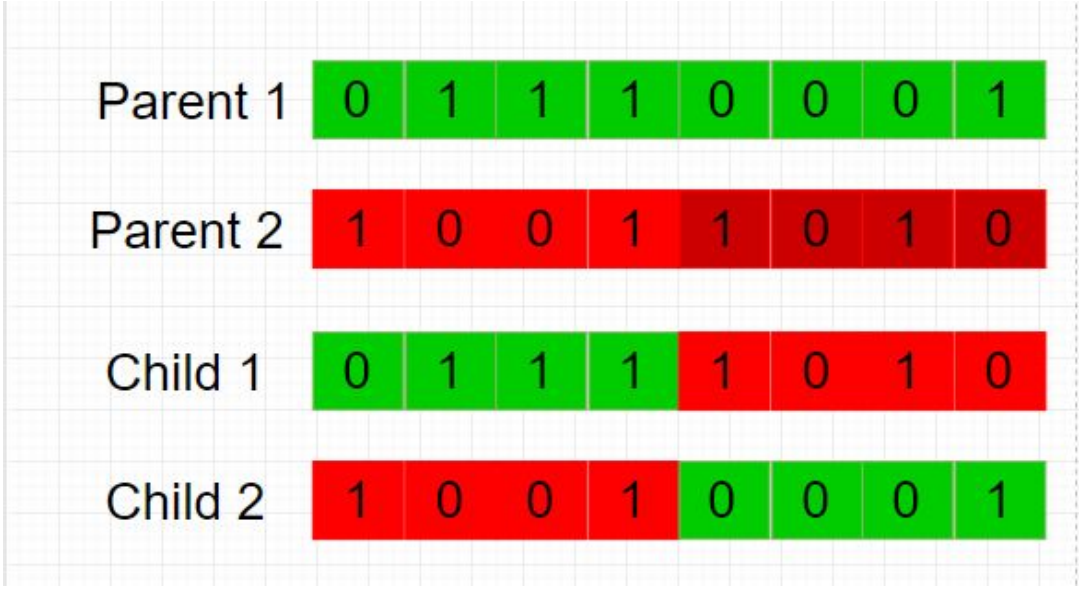

Imagem 3: Exemplo de Crossing Over.

2.3.6. Fase de Mutação

De maneira a garantir diversidade na população, a fase de Mutação surge para permitir a alteração de um bit em qualquer indivíduo da população, com uma probabilidade bastante reduzida. Esta alteração pode ocorrer em qualquer bit da população. São gerados números aleatórios X vezes, onde X = tamanho da população * tamanho do cromossoma. Caso o número aleatório gerado seja inferior à probabilidade indicada, então o bit correspondente ao número da iteração iteração é alterado.

Bit position	Random number
112	0.000213
349	0.009945
418	0.008809
429	0.005425
602	0.002836

Imagem 4: Exemplo de mutação. Retirado dos slides do professor Eugénio Oliveira.(1)

2.3.7. Condições de Paragem

O algoritmo é calculado N vezes, que é um valor determinado pelo utilizador.

Após a N-ésima vez, o algoritmo para e devolve a melhor conferência obtida até a altura.

3. Desenvolvimento

De maneira a obter uma rápida implementação, o programa foi desenvolvido em Java, tendo sido utilizada a versão "10.0.1". Utilizando o Eclipse como IDE, o ambiente de desenvolvimento variou entre Windows e MacOS.

Para conseguir ultrapassar algumas dificuldades, o código fonte da Framework Watchmaker(4) foi analisado.

3.1. Estrutura da Aplicação

A aplicação foi estruturada em alguns ficheiros java, separando o algoritmo genético do parse de um indivíduo da população. Existe ainda um ficheiro Utilities.java que contém todas as variáveis utilizadas pelo decorrer do programa, incluindo as variáveis fornecidas pelo utilizador.

Também é fornecido alguns ficheiros com dados para testes. Assim, o utilizador pode alterar estes ficheiros conforme quiser.

Por fim, ainda foi implementada uma interface gráfica, com recurso do Java Swing, para facilitar a interação do utilizador com o programa, tornando-a mais esteticamente apelativa e permitindo ao utilizador introduzir dados com maior facilidade.

3.2. Diagrama de Classes

Em baixo, segue o diagrama de classes que representa o programa e as interações inter-classes.

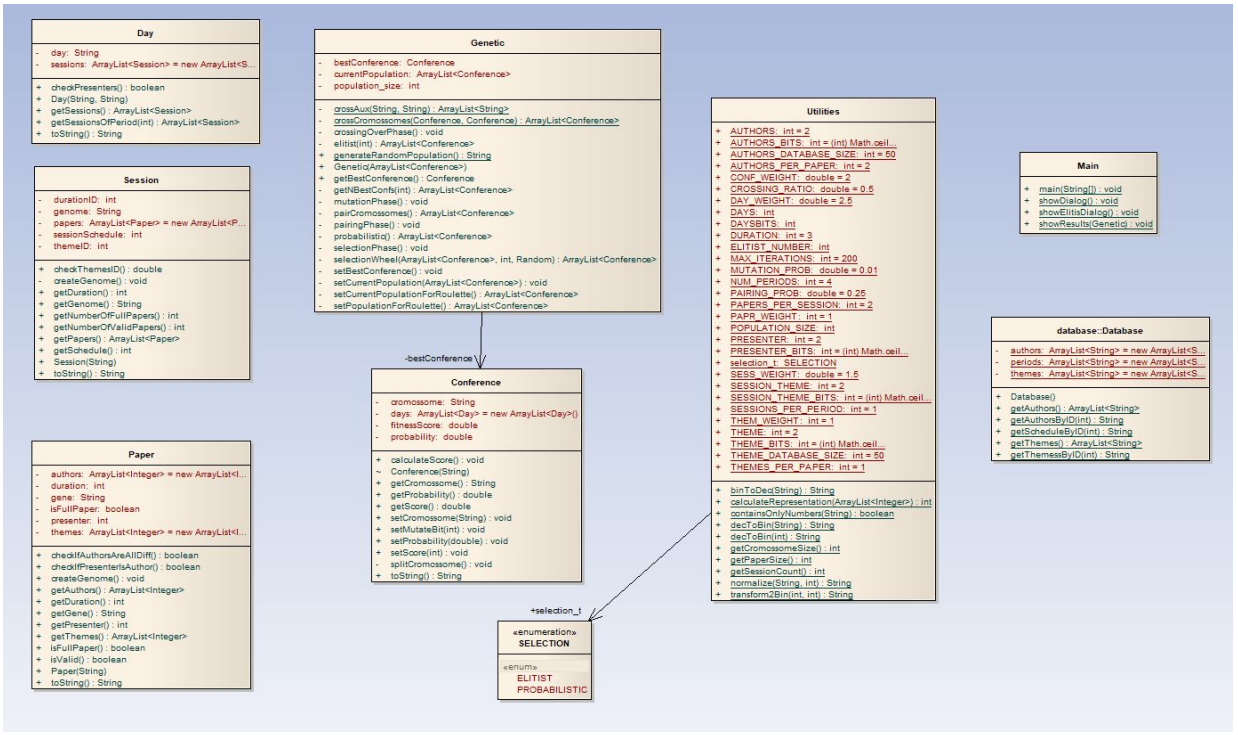

Imagem 5: Diagrama de Classes

3.3. Detalhes da implementação

De maneira a conseguir obter um tempo de computação o mais reduzido possível, as funções foram implementadas com a menor complexidade temporal conseguida.

O quadro seguinte mostra a complexidade temporal obtida nas fases do algoritmo.

Fase	**Complexidade Temporal**
Seleção	*O(n) -> probabilístico* *O(n*logn + n) -> elitista*
Emparelhamento	*O(n)*
Cruzamento	*O(n^3)*
Mutação	*O(n)*

4. Experiências

Resultados

Os primeiros testes serão efetuadas com as configurações padrões do programa. Essas configurações foram pensadas para deixar o mais próximo de uma situação real de uma conferência. Sendo estas 4 períodos por dia durante 3 dias, dois autores por paper, dois temas por paper e dois temas da sessão.

Também atribuímos pesos à certas ações de nossos programa, de forma a gerar os melhores resultados e scores melhores quando certas características ocorrem. Atribuímos pesos maiores para a diferença de horários nas sessões do dia e na diferença de horários nas sessões somadas de todos os dias para garantir horários equilibrados. Para outras características atribuímos multiplicadores menores pois elas não influenciam tanto na seleção do melhor indivíduo.

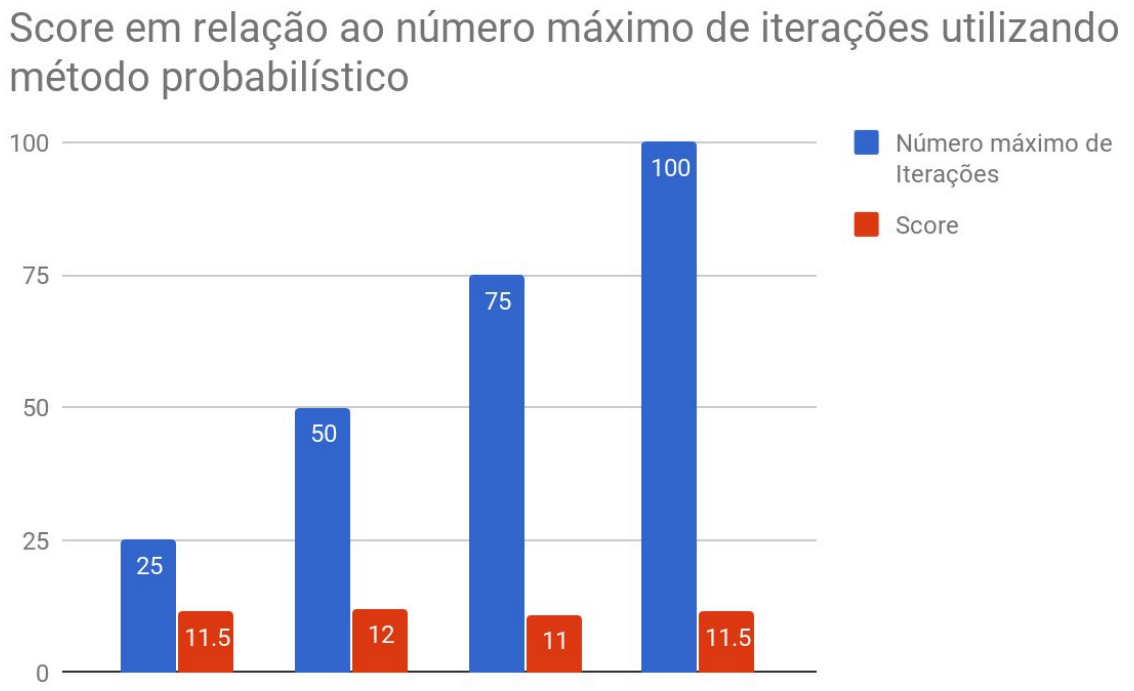

Imagem 7: Análise do score no algoritmo probabilístico em relação ao número máximo de iterações

Os primeiros testes efetuados buscam analisar a relação do número máximo de iterações com o score. Alisando os dados acima, pode-se perceber que que após a execução do algoritmo o score varia muito pouco em relação ao número máximo de iterações, produzindo até resultados inconsistentes pois quando o número de iterações aumenta, o score cresce coerentemente.

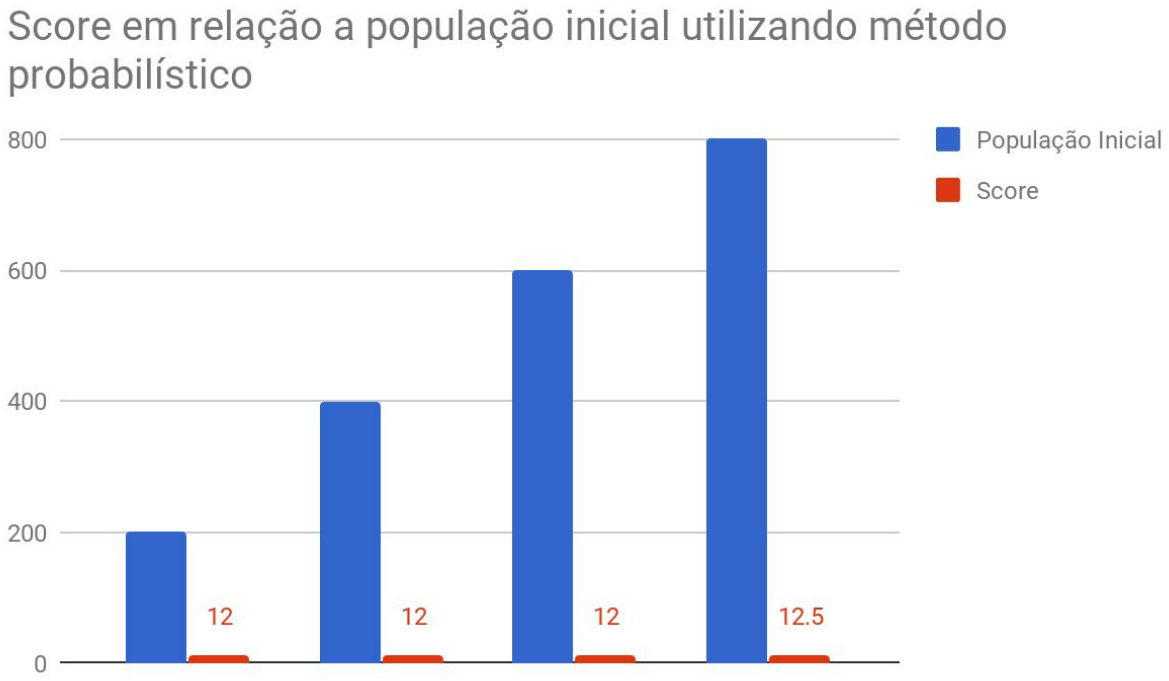

Imagem 8: Análise do score no algoritmo probabilístico em relação à população inicial

Nos segundos testes, verificamos o score em relação a população inicial. Podemos perceber uma consistência nos dados, onde o score se mantém equilibrado nas primeiras execuções porém sofre um leve aumento quando a população cresce para 800. Então podemos dizer que existe uma melhora no score quando a população inicial aumenta.

Imagem 9: Análise do score no algoritmo elitista em relação à população inicial.

É possível verificar que o score utilizando o método elitista permanece igual apesar do aumento recorrente da população inicial. Isto confirma que mesmo com populações menores o algoritmo já consegue atingir valores de score quase ótimos.

Imagem 10: Análise do score no algoritmo elitista em relação ao número máximo de iterações.

Nos últimos testes, pode-se verificar que aumentando gradualmente o número máximo de iterações se vê um aumento significativo do score. Finalizando assim a perspectiva de que o aumento de iterações lidera a melhores scores, neste caso de 12 na iteração final.

5. Conclusões

O desenvolvimento do projeto foi, em grande parte, despendido na implementação da função de avaliação e na decisão da representação de um indivíduo.

A utilização de algoritmos genéticos fornece um ganho significativo no que diz respeito ao tempo gasto para se solucionar o problema, pois é possível, em apenas algumas iterações, remover indivíduos que representam soluções inválidas e cruzar os melhores, formando rapidamente uma solução ideal, sendo pertinentes num espaço de pesquisa elevado.

Por conseguinte, os algoritmos genéticos calculam sempre uma boa solução, garantidamente melhor do que a inicial. Num ponto negativo, esta solução calculada pode não ser a solução ótima, mas não deixa de ser uma boa aproximação.

6. Melhoramentos

A condição de paragem podia ser aprimorada, fazendo uma verificação do *score* do melhor indivíduo: se este não aumentar passadas *n* iterações, então esse será considerado o melhor indivíduo e o algoritmo para.

Algumas validações mais avançadas podiam ser efetuadas para melhorar o score.

7. Recursos

7.1. Bibliografia

1. Oliveira, Eugénio: Métodos de Resolução de Problemas e Algoritmos para a Evolução;
2. https://bcc.ime.usp.br/tccs/2003/anselmo/node12.html
3. Watchmaker Framework

7.2. Software

1. Eclipse IDE for Java Developers
2. Java Swing
3. Windows e Mac
4. Github

7.3. Distribuição de Trabalho

Anderson Rogério da Silva Gralha - 1/3

António Miguel Silva Pereira - 1/3

Vitor Emanuel Fernandes Magalhães - 1/3

Toda a equipa trabalhou equitativamente, portanto a distribuição é igual.

8. Apêndice

8.1. Manual de Utilizador

Para correr o programa, basta importar o projeto para Eclipse e correr a partir deste último.

No início da execução, serão pedidos alguns valores, necessários para a execução do programa. Após o preenchimento destes valores, o programa correrá o algoritmo e, após terminar, será mostrado a melhor conferência e a sua identificação.

No ficheiro *Utilities.java*, existem alguns valores que poderão ser alterados para experimentar uma execução mais avançada. Porém, estas alterações poderão afetar negativamente a execução do programa, originando resultados inesperados.

Juntamente com o código fonte, é também fornecida uma documentação do projeto.