Relatório

Gregory Oliveira da Silva
gos@icomp.ufam.edu.br

1. Introdução

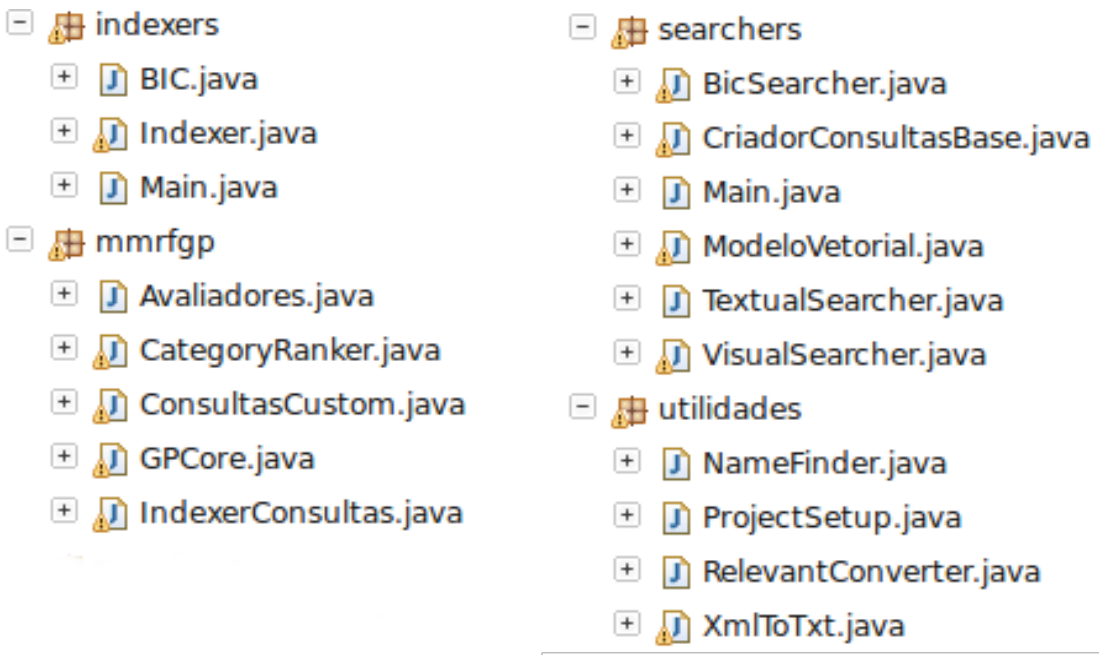
Este trabalho utiliza Programação Genética (GP) para aprender uma função de ranking que utilize valores de similaridades obtidos por diferentes fontes de informação. A base de imagens utilizada neste trabalho contem 23155 imagens de produtos como roupas e acessórios, e fornece também informações textuais sobre as mesmas, como descrição, preço, categoria e outros.

A função de aptidão a ser otimizada pelo GP consiste em uma avaliação de ranks obtidos por 50 imagens de consultas, onde os resultados relevantes já são conhecidos. O processo de recuperação de informação usado para cada consulta consiste na expansão de rank por busca multi-modal, onde ranks obtidos por CBIR são expandidos usando informação textual.

O objetivo deste trabalho é descobrir uma combinação de evidências visuais e textuais que obtenha um resultado ótimo pela função de aptidão. Para isso são gerados diversas possíveis soluções aleatoriamente, as quais evoluem com o passar das gerações passando por critérios de seleção que simulam a seleção natural de espécies.

2. Estrutura do código

O código foi implementado em Java, utilizando o framework JGAP para a implementação do GP, e o framework LIRE para a implementação das buscas visuais. A estrutura do projeto é mostrada a seguir:



Primeiramente, todos os caminhos para diretórios necessários para o funcionamento do programa são especificados na classe *ProjectSetup* no pacote *utilidades*. Este pacote contém classes responsáveis por alguns pré-processamentos, como a conversão de arquivos XML para Txt.

O pacote *indexers* contém classes responsáveis por processar a base de imagens e extrair os

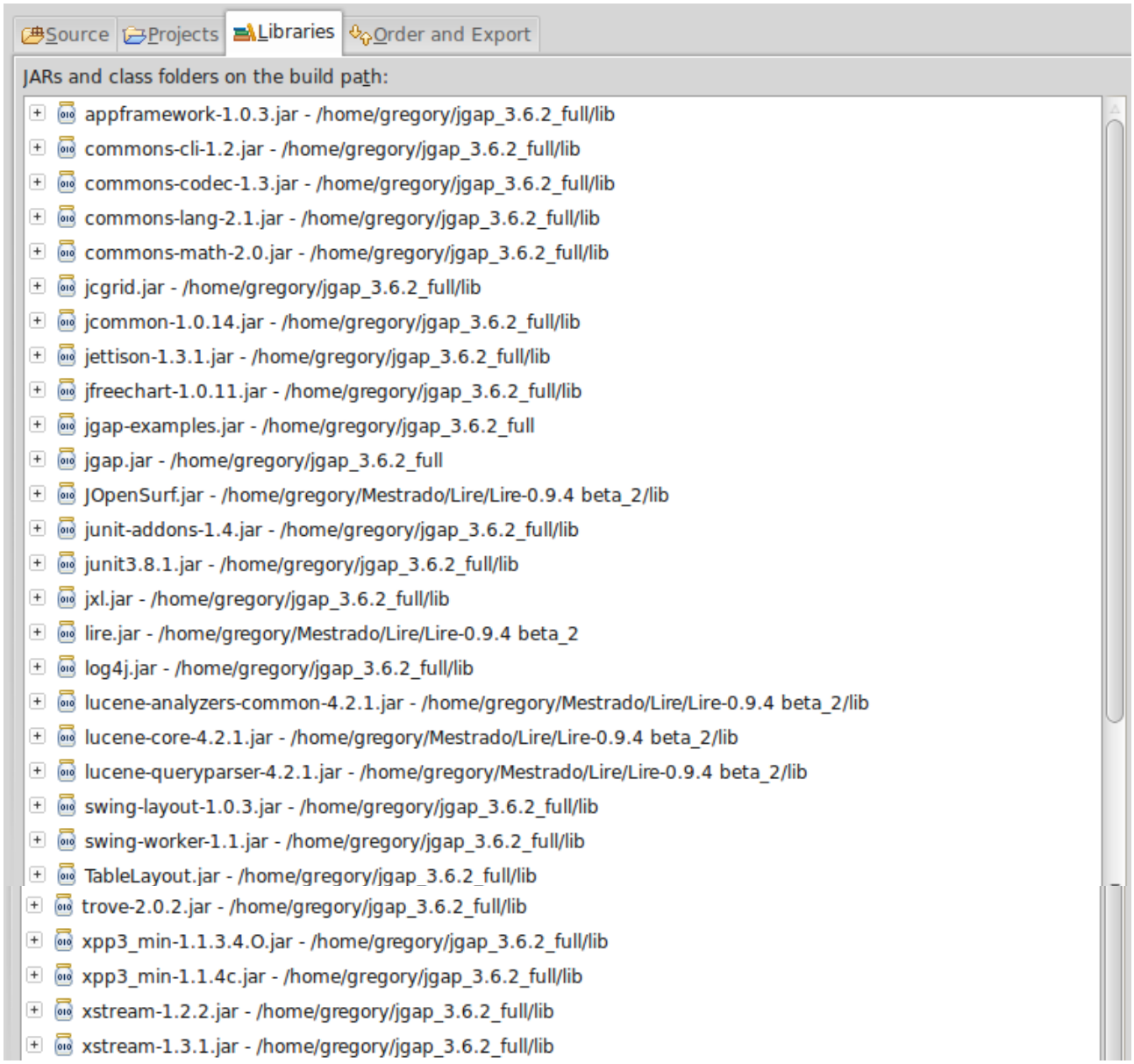
vetores de características de cada uma, estruturando-os de uma forma que possam ser usados para a recuperação de imagens visual. O pacote *searchers* contém classes que usam os índices para gerar os ranks com base em buscas visuais e textuais. O pacote *mmrfgp* (*multi-modal relevance feedback genetic programming*) contém o código referente à implementação do método de GP, junto com classes que implementam funções auxiliares, como avaliadores de rank e outros algoritmos.

3. Executando o código

Para executar o código é necessário baixar os seguintes arquivos:

- Eclipse: https://www.eclipse.org/downloads/
- JGAP: http://sourceforge.net/projects/jgap/files/OldFiles/jgap-0.1.zip/download
- LIRe v0.9.5: http://www.itec.uni-klu.ac.at/~mlux/lire-release/

Adicione os seguintes arquivos JAR no buildPath do projeto no Eclipse:



Por questões de desempenho, todas as buscas para cada imagem de consulta, utilizando cada descritor, são pré computadas e seus resultados são salvos em arquivos de texto. O mesmo ocorre

para as expansões dos ranks. Os arquivos XML da base de dados também são pré processados. Para gerar esses arquivos é necessário seguir estes passos:

1. Configure a classe *ProjectSetup* com os caminhos dos arquivos da base de imagens;
2. Compile e execute as classes *RelevantConverter* e *XmlToTxt*;
3. Compile e execute a classe *Main* do pacote *indexers*. Desta forma são gerados os vetores de características das imagens. São executados 9 algoritmos de descrição (classes *indexer* e *BIC*) para cada uma das 23155 imagens, por isso esse processo pode demorar algumas horas;
4. Compile e execute a classe *Main* do pacote *searchers*. Serão gerados ranks para cada descritor, totalizando 50 x 9 ranks, através da classe *VisualSearcher* e *BicSearcher.* Em seguida esses ranks são expandidos (classes *TextualSearcher* e *ModeloVetorial*) utilizando as informações textuais presentes nas imagens do topo do rank utilizando k imagens, onde k varia entre 1, 5, 10 e 20. Desta forma totalizam 50 x 9 x 4 ranks expandidos;
5. Compile e execute a classe *GpCore* do pacote *mmrfgp*. Nesta classe a função de aptidão consiste em, dada uma imagem de consulta, combinar todos os ranks obtidos para a mesma utilizando uma função de ranking a ser avaliada, e em seguida avaliar o rank formado usando P@10 (classe *Avaliadores*). É calculada a média das precisões para as 50 imagens de consulta, e este valor é dado como a aptidão da função de ranking.

4. Saída de dados

Ao executar o código, são impressos no console informações sobre sua execução, e em seguida os melhores indivíduos de cada geração. Um exemplo por ser obsercado a seguir:

```
[JGAP][11:02:42] INFO GPGenotype - Creating initial population
[JGAP][11:02:42] INFO GPGenotype - Mem free: 240.6 MB
[JGAP][11:02:42] INFO GPPopulation - Prototype program set
[JGAP][11:02:42] INFO GPGenotype - Mem free after creating population: 240.6 MB
[JGAP][11:02:42] INFO GPGenotype - Your configuration does not contain unused commands, this is good
[JGAP][11:02:42] INFO GPGenotype - Evolving generation 0, memory free: 238.3 MB
[JGAP][14:57:18] INFO GPGenotype - Best solution fitness: 0.59
[JGAP][14:57:18] INFO GPGenotype - Best solution: max((min(CEDD_cat1, FCTH)), (FCTH_cat20 * ACC_cat5))
[JGAP][14:57:18] INFO GPGenotype - Depth of chrom: 2
[JGAP][22:00:57] INFO GPGenotype - Best solution fitness: 0.64
[JGAP][22:00:57] INFO GPGenotype - Best solution: ((max(SIFT_text20, CLD_cat5)) + (max(GCH, FCTH_cat10))) + ((max((min(CEDD_cat1, FCTH)), (FCTH_cat20 * ACC_cat5))) + (GCH_cat1 * FCTH_text10))
[JGAP][22:00:57] INFO GPGenotype - Depth of chrom: 4
[JGAP][23:42:37] INFO GPGenotype - Best solution fitness: 0.67
[JGAP][23:42:37] INFO GPGenotype - Best solution: ((max(SIFT_text20, CLD_cat5)) + (ACC_cat5 * FCTH_cat5)) + JCD
[JGAP][23:42:37] INFO GPGenotype - Depth of chrom: 3
[JGAP][03:05:24] INFO GPGenotype - Best solution fitness: 0.71
[JGAP][03:05:24] INFO GPGenotype - Best solution: ((max(SIFT_text20, CLD_cat5)) + CEDD) + ((max((min(CEDD_cat1, FCTH)), (FCTH_cat20 * ACC_cat5))) + (GCH_cat1 * FCTH_text10))
```