West University of Timisoara

Faculty of Mathematics and Computer Science

Study Program:
Computer Science in English

Bachelor Thesis

COORDINATOR:

Conf. Dr. Marc Eduard Frîncu

GRADUATE:

Bulz Gabriel

Timişoara
2018

West University of Timisoara

Faculty of Mathematics and Computer Science

Study Program:
Computer Science in English

Prezicerea zonelor inundabile folosind imagini satelitare

COORDINATOR:

Conf. Dr. Marc Eduard Frîncu

GRADUATE:

Bulz Gabriel

Timişoara
2018

Abstract

Floods are without doubt the most devastating natural disasters, striking numerous regions in the world each year. During the last decades due the increased frequency of heavy rain and a continuously increasing concentration of population near water regions a lot of assets and lives were lost.

That is the reason why a system that can create a flood forecasting is

To process the resulted images we will use a library in python called GDAL which can handle that specific type of files (.tif)

In the first part we will try to identify the areas nearby the rivers and lakes (because that areas are more likely to be flooded), and in the second part we will try to analyze the images based on the height of each section. The water from the rain will gather in the lower areas, so we can presume that that areas are likely to hold water.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Floods are without doubt the most devastating natural disasters, striking numerous regions in the world each year. During the last decades due the increased frequency of heavy rain and a continuously increasing concentration of population near water regions a lot of assets and lives were lost.

In general, less developed countries are the most vulnerable to floods, causing damages that significantly affect the national GDP. At country and community levels important initiatives have and are being devoted to implement appropriate countermeasures, both structural and non-structural, aiming to alleviate the persistent threats of water-related disasters.

1.2 Our Contribution

1.3 Thesis Structure

Chapter 2

Technologies Used

2.1 Programming languages and Frameworks

Programming language: Java

Java is popular high-level programming language that is object-oriented and it is designed to have as few implementation dependencies as possible. It was created in 1995 by Sun Microsystems and now it is owned by Oracle.

This project was developed using Java 1.8 with two external libraries i.e.: Gdal and jai-imageio-core 1.4.0. Any version below 1.6 will now work as intended because of jai-imageio-core library. This library was used to process TIFF images from satellite.

At first we intended to use Python as the heavy lifting programming language because of the more relaxed syntactic structure, but in the end we chose Java because of its parallel programming capabilities. Usually when we have to work with satellite images we should take into account the fact that the images can be really large, like 8161x7211 pixels in size covering over 589.000.000 km^2, and when we have to process more that one image of this kind the processing time becomes very important.

Python will become much slower in this area because of the Global Interpreter Lock or GIL (a mutex, or a lock, that allows only one thread to hold the control of the Python interpreter). This means that only one thread can be in a state of execution at any point in time. The impact of the GIL isn't visible to developers who execute single-threaded programs, but it can be a performance bottleneck in CPU-bound and multi-threaded code.

In the following image we can take a look over the processing time between the Java's multi-threaded system and the single-threaded Python's system

Multi Thread

```
For 8 images with multi threadin it took:
24728
```

Single Thread

```
For 8 images with single threadin it took:
39675
```

Programming language: Python

Python is a interpreted, high-level, open source programming language that was made to have a relaxed syntactic structure and to be more easy to read. It supports both functional and object oriented programming and its mainly targeted to a fast development and easy to maintain code.

Python had played a major role in the project because of its relaxed syntactic structure and it was use to create the server part of the application, using Flask (flask is a microframework for python based applications).

After the server receives a set of satellite images from the user, a Java process is started by the server to solve the request. The java files are complied when the server is started for the first time and then for every request a java process is started. All this part was handled using the python's "subprocess" library

We will take a short look on how the python server compiles the java files and how a process is started, and we will discuss in more details in the next chapters.

Here we can take a short look on how the java files are compiled

```python
import subprocess

def compile_java_files():
    """
        compile all java files that are used for image processing
        and link all libs
    """
    java_files = JAVA_FILES_PATH + '\\*java'
    java_libs = JAVA_LIBS_PATH + '\\*'
    subprocess.Popen(['javac', '-d', JAVA_OUT_COMPILED_CLASSES,
                      '-sourcepath', JAVA_SOURCEPATH_CLASSES,
                      '-cp', java_libs, java_files], shell=True, stdout=True)
```

Here we can take a look on how a process is started

```
def process_files(path_files, result_directory):
    """
        run a java process that will solve the request
        param : path_files - path to the folder where images have been unziped
        param : result_directory
        return : 1 - if the process have failed
                 0 - if the process had succed
    """
    java_libs = JAVA_LIBS_PATH + '\\*'
    compiled_libs_and_classes = JAVA_OUT_COMPILED_CLASSES + ';' + java_libs
    java_main_class = JAVA_PACKAGE_NAME + '.' + JAVA_MAIN_CLASS
    cmd = ['java', '-cp', compiled_libs_and_classes,
           java_main_class, path_files, result_directory]
    stdin = PIPE
    stdout = PIPE
    stderr = STDOUT
    proc = subprocess.Popen(cmd, stdin=PIPE, stdout=PIPE, stderr=STDOUT, shell=True)
    stdout, stderr = proc.communicate()
    output = str(stdout).replace('\\r','').split('\\n')
    final_output = str(output[0]) + str(output[1]) + str(output[2])
    correct_output = JAVA_OUTPUT_DETECT_CLASS + JAVA_OUTPUT_PREDICT_CLASS + JAVA_OPTIONS

    if (final_output == correct_output):
        return 0
    return 1
```

Features

- Coding assistance and analysis, with code completion, syntax and error highlighting, linter integration, and quick fixes
- Project and code navigation: specialized project views, file structure views and quick jumping between files, classes, methods and usages
- Python refactoring: including rename, extract method, introduce variable, introduce constant, pull up, push down and others
- Integrated Python debugger
- Integrated unit testing, with line-by-line code coverage
- Version control integration: unified user interface for Mercurial, Git, Subversion, Perforce and CVS with changelists and merge

2.2 Python Programming Language

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. In July 2018, Van Rossum stepped down as the leader in the language community after 30 years.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation.

2.3 Gdal

The Geospatial Data Abstraction Library (GDAL) is a computer software library for reading and writing raster and vector geospatial data formats, and is released under the permissive X/MIT style free software license by the Open Source Geospatial Foundation. As a library, it presents a single abstract data model to the calling application for all supported formats. It may also be built with a variety of useful command line interface utilities for data translation and processing. Projections and transformations are supported by the PROJ.4 library.

The related OGR library (OGR Simple Features Library), which is part of the GDAL source tree, provides a similar ability for simple features vector graphics data.

GDAL was developed mainly by Frank Warmerdam until the release of version 1.3.2, when maintenance was officially transferred to the GDAL/OGR Project Management Committee under the Open Source Geospatial Foundation.

GDAL/OGR is considered a major free software project for its "extensive capabilities of data exchange" and also in the commercial GIS community due to its widespread use and comprehensive set of functionalities.

How to install Gdal on a Windows machine

- The user will need to install a version of Python
- The user will need to install the Miniconda packages Conda (recommended Conda - a more complex version of miniconda)
- The user will need to install the Gdal Api's using the following commands : conda install -c conda-forge gdal or conda install -c conda-forge/label/broken gdal

2.4 Qgis

QGIS (previously known as Quantum GIS) is a free and open-source cross-platform desktop geographic information system (GIS) application that supports viewing, editing, and analysis of geospatial data.

Functionality

QGIS functions as geographic information system (GIS) software, allowing users to analyze and edit spatial information, in addition to composing and exporting graphical maps. QGIS supports both raster and vector layers; vector data is stored as either point, line, or polygon features. Multiple formats of raster images are supported, and the software can georeference images.

QGIS supports shapefiles, coverages, personal geodatabases, dxf, MapInfo, PostGIS, and other formats. Web services, including Web Map Service and Web Feature Service, are also supported to allow use of data from external sources.

QGIS integrates with other open-source GIS packages, including PostGIS, GRASS GIS, and MapServer. Plugins written in Python or C++ extend QGIS's capabilities. Plugins can geocode using the Google Geocoding API, perform geoprocessing functions similar to those of the standard tools found in ArcGIS, and interface with PostgreSQL/-PostGIS, SpatiaLite and MySQL databases.

How to install Qgis

- The user will need to download and install the Qgis Standalone Installer corresponding to the operating system 32/64
- Through the OSGeo4W shell's the user will be able to realise fast operations on the data from the satellites

Chapter 3

The application

3.1 Qgis introduction

3.2 Python GDAL introduction

3.3 Functional description

3.4 The user interface

3.5 Main use cases

3.6 Implementation details

3.7 Application specification - user's point of view

User's point of view

This app should be able to deliver a marked zone on the land area(provided by the user), based on the probabilities of that zone to be floodable. (ex: red for high probability, green for none)

The user will have to insert a set of images provided by sentinel 1,2 or 3, or a .tif representation of the area which he wants to test. The .tif set of images should include a infrared picture as well, because that is the way in which the water can be detected. The satellites usually use infrared scanners so this would not be a problem. The infrared images are useful because the water will absorb the infrared laser, so the areas covered with water are usually black. This is how the program will find the water areas, and will be able to detect the zones which have a higher chance of being flooded. Depending on the progress of the program, we will try to offer a GUI for the user in which he/she will be able to insert the coordinates of the area under the observation, and we will try to download the images based on their coordinated using a API for sentinel satellites.

The resulted image will give the user a sense of which areas are more likely of being flooded during a heavy rain period.

3.8 Application specification - programmer's point of view

User's point of view and application skeleton

This app should be able to deliver a marked zone on the land area(provided by the user), based on the probabilities of that zone to be floodable. (ex: red for high probability, green for none)

The app will receive a set of .tif images, (provided by the user) from Landsat, sentinel 1,2 or 3, one image will be in infrared and the other one will be composed from the green wave channel.

The way that we detect water is by combining the NIF(infrared) with green band resulting NDWI (Normalized Difference Water Index). NDWI is a remote sensing based indicator sensitive to the change in the water content.NDWI is computed using the near infrared (NIR–MODIS band 2) and the short wave infrared (Green band) reflectance's.

The formula will be applied as following for each pixel:

$$NDWI/perpixel = \frac{Xgreen - Xnir}{Xgreen + Xnir}$$

If the resulted pixel has a value bigger that 0.45 we classify that specific pixel as a pixel that contains/is water (lake, river, ocean, etc), and all the pixels below that value as non-water pixels (land). We will color the water pixels in white and the other ones in black.

Using this method we will obtain all the water content of a picture, and we can mark the areas which have a high risk of being flooded(the areas near the water).

Later the user will be able to add a set of topographic images and determine the exact areas that will be flooded, by combining the results from NDWI water area detection with the topographic height of the map.

Chapter 4

Conclusions

4.1 Performance Evaluation

4.2 Future Development

Chapter 5

Bibliography

5.1 Bibliography here