Intelligent Systems Lab 23/24

Domain of the problem

The **Instituto Geográfico Nacional** (IGN) participates in the National Aerial Orthophotography Plan (PNOA) which is a joint project between the Government and the Regional Governments that seeks to obtain digital aerial orthophotographs of the whole of Spain, updated every 3 years, and has been incorporating LiDAR technology since 2009.

In 2009, after pilot tests and the need for greater precision in digital terrain models, the PNOA-LiDAR project arose, which seeks to map Spain in a 6-year cycle with LiDAR data. Three coverages will be carried out: 2009-2015, 2015-2021 and 2023 onwards (2022 in Catalonia). The third coverage will have a higher density of points (5 p/m^2) to improve resolution and uses.

1. Geographical information: La Gomera

The geographical area of study for this lab is the island of La Gomera,, belonging to the Canary Islands, and whose geographical information can be found in the resources of the Modelo Digital del Terreno del PNOA.

2. Hierarchical files. HDF5 format

In order to be able to work with the values of all the files efficiently, a hierarchical file has been built to store all the information. The format used will be the **Hierarchical Data Format (HDF5)**.

The following is a brief introduction to this format and an indication of the characteristics that have been considered for storing all the data of the Modelo Digital del Terreno of the island of La Gomera. The complete documentation and reference on the HDF5 format can be found on The HDF Group.

Introduction to HDF5

Here we will explore what HDF5 files are and how they are organised, providing a basic understanding of this file format used in a variety of areas such as scientific research, engineering, simulation, data analysis, etc.

What are HDF5 files?

HDF5 files, or Hierarchical Data Format 5, are a versatile solution for storing and managing large volumes of data. Designed to address the need to store complex, multidimensional data, HDF5 files have gained popularity in disciplines where organisation and efficient access to data are essential.

HDF5 Structure:

HDF5 files follow a hierarchical structure, similar to the organisation of folders and files in a traditional file system. Here are the key elements of an HDF5 file structure:

- **Groups**: Equivalent to folders, groups allow data to be organised logically. They can contain data sets and other groups, forming a hierarchy.
- **Datasets**: These are multidimensional arrays that store the actual data. Each dataset has a unique name and is associated with a data type that defines the format of the elements in the array.
- **Attributes**: This metadata provides additional information about groups and datasets. They can describe units, authors, dates and more.
- **Custom Datatypes**: HDF5 files allow custom data types to be defined, making it possible to handle complex data structures.
- **Dataspace**: Define the distribution of data in multidimensional sets. They can be simple or complex, depending on the needs.
- **Property List**: a collection of parameters (some permanent and some transient) controlling options in the library
- **Links**: Links allow you to create references to groups or sets of data in different parts of the file or in other HDF5 files.

In short, HDF5 files offer a flexible structure for efficiently storing and organising data. They are ideal for situations where complex, multidimensional data is handled, providing powerful tools for data management and analysis in a variety of areas. In the next sections, we will explore each of these components in more depth, giving you a solid understanding of how to work with HDF5 files.

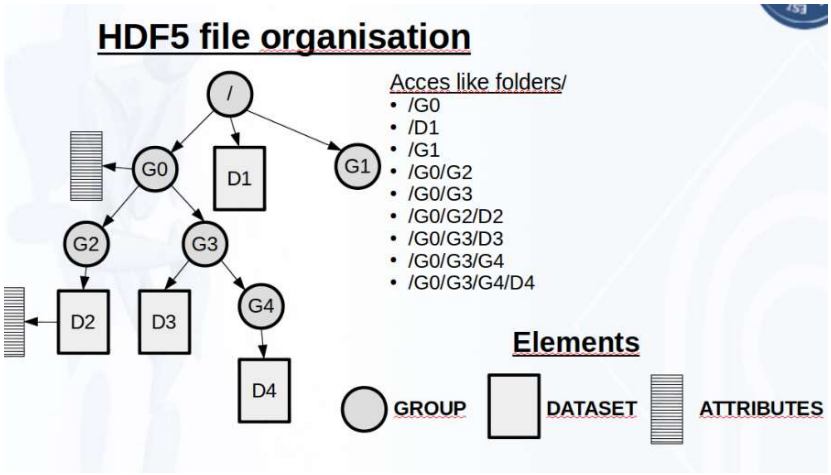


Figure 3. Esquema de los elementos básicos del API de HDF5

Five datasets have been created from the initial root group '/' named with the distinctive numbers in the names of each ASC file. For example '/1095-1-2' would give access to the data grid of the file MDT02-REGCAN95-HU28-1095-1-2-COB2.asc. To reduce the file size, these data grids have been stored in gzip format.

Each dataset has the following attributes derived from the attributes of its ASC file:

- ***xinf****: minimum X-UMT coordinate of the dataset.*
- ***yinf:*** *minimum Y-UMT coordinate of the dataset.*
- ***xsup****: maximum X-UMT coordinate of the dataset.*
- ***ysup:*** *maximum Y-UMT coordinate of the dataset.*
- ***cellsize:*** *The size of one side of a cell. In this case with value always 2.*
- ***nodata_value:*** *value without data. In this case always -99999.*

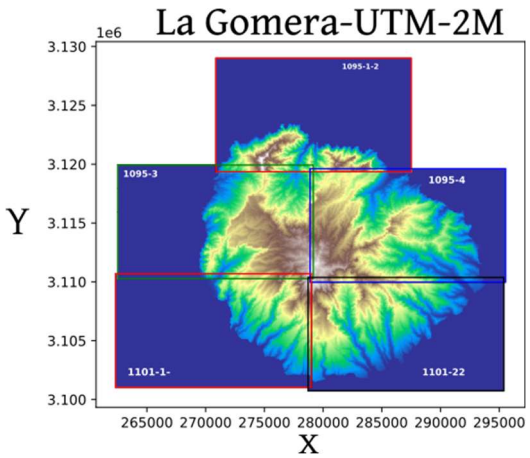


Figure 4. Picture with the content of the file LaGomera.hdf5

3. Results

Goal

The goal of Task 1 is the generation of a software artifact Map with the following characteristics and methods.

Map Artifact

Feature/Properties:

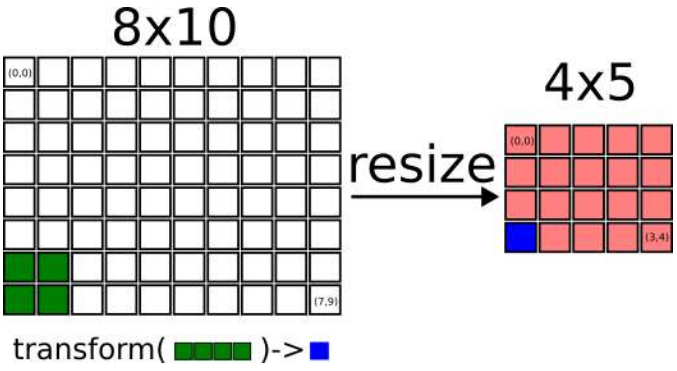
- ***Map.nodata_Value:float****. Value for grid cells that do not contain a valid value. If the map has more than one dataset this value must be the same for all.*
- ***Map.sizeCell:float****. Size in metres of a side of a cell in a dataset. If there are several datasets this value must be similar in all of them.*

- ***Map.upLeft**: (**max_Y:float,min_X:float)**. Coordinates Y-UMT, X-UMT at the top left-hand corner of the map.*
- ***Map.downRight**->**(min_Y:float,max_X:float)**. Coordinates Y-UMT, X-UMT at the bottom right-hand corner of the map.*
- ***Map.dim**->**(rows:integer,columns:integer)**. Overall dimensions of the Grid.*
- ***Map.filename**:**string**. Name of the hdf5 file.*
- ***Map.f**: File with format hdf5.*

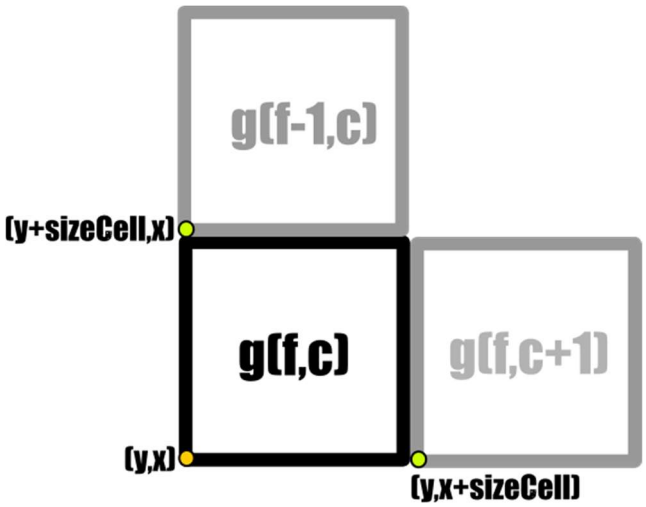
Methods/Actions

- **Map(filename:string)**: Having as an input/parameter the name of an hdf5 file of the indicated type, as in the example of **LaGomera.hdf5**, a Map is created with this information.n.
- **Map.umt_YX(y,x:float)->float**. Given the coordinates Y-UMT and X-UMT it must return the corresponding value of the grid cell corresponding to the position of those coordinates. If no value exists in those coordinates it will return the value Mapa.nodata_Value. (See below **Explanatory note** on belonging a point to a cell).
- **Map.resize(factor:integer, transform:f(cells:array[float]):float,name:string)->Map**. Let be a map with a grid with dimensions Map.dim()=(nrows,ncolumns), the method **resize** creates **a new map** with a new **name** and a new grid with dimensions (nrows/factor ncolumns/factor) and a new cell size of cellsize*factor.

    - The value of the new dimensions is rounded up to the nearest integer. For example a value of 4.35 would be rounded to the value of 5.
    - The **value of the new cell** is calculated by the **transform** function that receives the cells of the original grid as a parameter.
    - For example, taking Figure 5 and applying **resize(2,func,'NewMap')** we start from a Map that has a 8x10 grid and obtain a Map called **NewMap** with a **4x5 grid**. The value of each new cell is obtained by applying the function **func to 4 cells (2x2)** of the original grid.



Note: As it can be observed in the next figure, for any point, UMT(y',x'), to belong to the cell ***g(f,c)*** it must be satisfied that ***y<=y'<y+sizeCell*** and ***x<=x'<x+sizeCell***.



Problem (Definition)

As previously indicated, the goal of this Lab is to obtain routes starting from a specific place (geographical point UMT (Yo,Xo)) and arriving at a destination UMT (Yd,Xd), by means of a set of valid movements.

Before finding how to solve the problem, we have to concretise it and define it perfectly. For this, we have to specify:

The state space: States + Actions

- Initial State.
- Objective function.

1. State Space

STATE

A state is ***the description of a situation in a specific moment in time***. The most typical way to represent a state is to determine which elements of the problem/environment/agent are relevant, identify them as ***variables***, and determine its ***domains*** as the potential values it can take.

Once we have established the variables and its domains, ***a state is the specific assignation of values to those variables***.

Variables and Domains

In our case we can identify a specific situation:

- ***Where am I.*** Geographic location where I am currently situated. The domain of this variable would be the set of points of the Map.

Once we have defined the state, we can have a representation to show it as a string of characters as follows:

"(3100747,262333)" which generically will be a string formed as:

"**(<Y_UMT>,<X_UMT>)**".

To maintain uniformity, we will impose a restriction to the string that represents a state, It will not have blank spaces inside of it.

With the goal of having a unique state identifier, which can be used as key in any indexed structure, like a hash table or a dictionary, each state will have an ***id*** equivalent to the string representing it.

SUCCESOR FUNCTION

In the proposed problem, the only way to modify a state is through the action of moving from one point to another point. Considering that we start from a geographical point (Y,X) the valid displacement will have 2 variables:

- **Direction of the displacement**. Initially we will consider 4 possible directions in this order: North, East, South and West.
- **Longitude of the displacement**. Distance in meters of the displacement, measured in a factor of the cellsize of the Map used. For example a factor of 5 would be a distance of 5*cellsize meters. We will consider that this factor is kept constant in the problem.

For example, let's have the following state (Y,X) and a factor of 20, the set of adjacent states would be:

1. N, new state (Y+20*cellsize,X).
2. E, new state (Y,X+20*cellsize)
3. S, new state (Y-20*cellsize,X)
4. W, new state (Y,X-20*cellsize)

In order to maintain a uniform criterion for the order of the successors, it will be considered to be N,E,S and W.

To finish defining the necessary elements of a state space, we need to establish the cost of each action. For this purpose we will distinguish two different factors:

- **Length** as the length advanced in each action which is equivalent to factor*cellsize.
- **Height** as the slope saved which will be calculated as the absolute value of the difference in heights between the current state and the new state.

An ACT action for state (Y,X) will be given by the tuple:

({N|E|S|W},(Yd,Xd),(Length,Height)).

An action is considered valid if it verifies some kind of restriction on any of the elements that compose it, for example that it cannot perform the action if it exceeds a maximum Height or that the new state (Yd,Xd) is outside the Map or has no height value.

validate(ACT):Boolean

For this task, we consider an action Not Valid if the result of such action is a state outside of the map or a state with a cell value equal to *nodata_value*.

The SUCC function of the state (Y,X) shall be a list of valid actions: **SUCC((Y,X))=[<ACT1>,<ACT2>,...,]**

2. Problem

Once we have defined the state space with the states and the successor function, we only need to define an initial state and an objective function.

INITIAL STATE

Given a map, an initial state will be any valid state as previously described:

(Yo,Xo)

OBJECTIVE FUNCTION

The most direct way to recognise that we have reached a desired situation is when we have already reached the desired geographical position. This condition is easy to identify in a valid state:

GOAL((Y,X))=(Y==Yd) and (X==Xd)

3. Result

After finishing this second task, we will have an artefact **Problem** which will be defined through the name of the file of the map to be used, an initial valid state for this map, and the objective function.

To handle the state space we will create an artefact **State** that will encapsulate all the information relative to a state, as well as the successor function.

Search Algorithm

In this task we will develop the general **search algorithm**, for which we will need to develop the artefacts **frontier**, **tree node**, and set of **visited states**.

1. Frontier

The frontier must be a structure that stores the nodes of the search tree.

The characteristics we require are:

1. **Structure sorted** by the value of the nodes of the tree. If there are several nodes with *the same value they will be ordered by the ID's* of the nodes.
2. Due to efficiency, the frontier will have a **sorted insertion**, avoiding sorting after each insertion.

2. Nodes of the search tree

A search tree node will need to have the following elements:

- **ID**: An integer number. It is unique for each node and will be assigned in increasing order.
- **Parent**: A pointer of reference or value of the parent node who generated it.
- **State**: A state, this is the state of the node.
- **Value**: A real number. This value will be calculated through the different strategies seen in class.
- **Depth**: An integer number.
- **Cost**: A tuple (distance, maximum slope). The cost of the path to the node considering the total distance travelled and the maximum slope overcome.
- **Heuristic**: A real number, the value of the heuristic function for the state of the node.
- **Action**: A string, it will be the name of the successor which that node has generated.

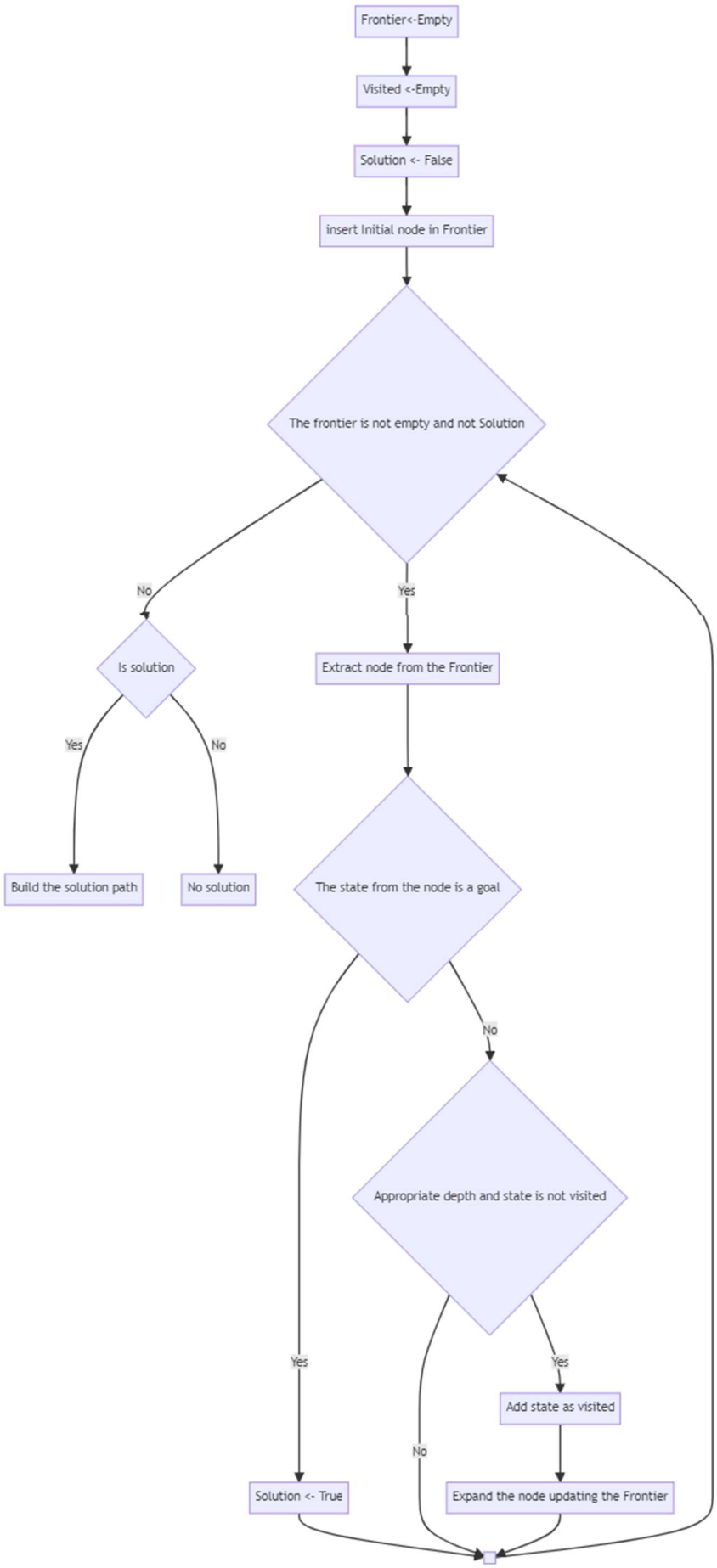
And a **path function** which will return a list of nodes from the root node to that specific node.

A node will have a textual representation as a string with the form:

[<ID>][<COST>,<STATE>,<ID_PARENT>,<ACTION>,<DEPTH>,<HEURISTIC>,<VALUE>]

Where <STATE> will be the ID of that state.

3. Algorithm


Frontier<-Empty
Visited <-Empty
Solution <- False
insert Initial node in Frontier
The frontier is not empty and not Solution
No
Yes
Is solution
Extract node from the Frontier
Yes
No
Build the solution path
No solution
The state from the node is a goal
No
Appropriate depth and state is not visited
Yes
Yes
Add state as visited
No
Solution <- True
Expand the node updating the Frontier

4. Expansion of a tree node

The expansion of a node is done through the *list of successors of its state*, maintaining the established order for said list.

The node.value will be assigned, according to the proposed strategy:

- **Breadth**: node.value <- node.depth
- **Depth**: node.value <- 1/(node.depth + 1)
- **Uniform Cost**: node.value <- node.cost.distance

The rest of the values of the node will be assigned as follows:

- **node.ID** <- Total nodes + 1
- **node.parent** <- parent
- **node.depth** <- parent.depth + 1
- **node.cost** <- As it is a tuple, it contains two values: **node.cost.distance** <- parent.cost.distance + cost.distance of considered successor and **node.cost.maxSlope** <- MAX(parent.cost.maxSlope, cost.slope of the considered successor).
- **node.state** <- next state of considered successor
- **node.action** <- name of the action

SET OF VISITED NODES (VISITED LIST OR CLOSED LIST)

The states will be defined by their ID and the set of visited nodes will prioritize efficiency for the operations:

1. **Belongs**: Checks if a state is in the set.
2. **Insertion**: Incorporates a state to the set.

Consider the use of *hash tables* with the ID of the states as keys.

5. Results

After completing this task we will have an algorithm to solve a problem, with a strategy and a maximum allowed depth, that must return the solution path.

Solution = SearchAlgorithm(problem, strategy, maximum_depth)

Heuristic and A* strategy

Once we have finished the implementation of the search algorithm, the final task of the laboratory consists in incorporating the Greedy and A* strategies.

We consider the use of two alternative heuristic functions: $\mathbf{H}_{\mathbf{euclidean}}$ and $\mathbf{H}_{\mathbf{manhattan}}$

1. Heuristic function (Euclidean)

A heuristic function, applied to a state, provides an estimate of the cost required to reach some goal state. In this case we will consider two distances as the cost estimate.

For the goal state of the problem (Dy,Dx) we will have that the values of the heuristic functions for any state (y,x) are calculated according to the following formulas:

- $\mathbf{H}_{\mathbf{euclidean}}\mathbf{((y,x))=sqrt((Dx-x)\text{^}2+(Dy-y)\text{^}2)}$
- $\mathbf{H}_{\mathbf{manhattan}}\mathbf{((y,x))=|Dx-x|+|Dy-y|}$

2. Results

The final result of this task is the incorporation of the Greedy and A* strategies to the Search Algorithm where two heuristic functions, $\mathbf{H}_{\mathbf{euclidean}}$ or $\mathbf{H}_{\mathbf{manhattan}}$, can be independently used.