

## **DAG, ShockGraph and GestureGraph**

DAG is the base class for all the graphs we want to match. Basically, the DAG-derived classes, such as ShockGraph and GestureGraph only extend the functionality of a DAG by defining a NodeDistance() function to measure the similarity of two nodes of the same type and a Read() function that computes a graph from a .ppm file or some other file format. In turns, DAG derives from the LEDA graph GRAPH<DAGNodePtr, double>, which allow us to use all the LEDA graph functions with any of our DAGs.

The DAG class is actually an abstract class that has a number of virtual and pure virtual functions. The design of this class is meant to allow us to deal only with pointers to DAG objects even though they may be actually pointing to DAG-derived classes. Thus, there are just a few lines of code for which is relevant to know the actual class of the DAG object that is being pointed to.

## **DAGDatabase**

This class represents both the object database and the TSV or index database. The goal of this class is to integrate a DAGSearchDatabase object (index db) and a DAGDBFile object (object db) under a unified framework and thus facilitate our work with these classes. For example, when an object database is open for matching, the creation time of the index database is checked to ensure that is newer than the modification time of the object database. Otherwise, the old index database is removed and a new one is created. The reason for this is that we need to know the maximum branching factor of every DAG when creating the index database, and so it's not always possible to update the index database each time the object database is updated.

## **DAGSearchDatabase**

The DAGSearchDatabase is a nearest-neighbour database that contains the TSVs for all the nodes of all the graphs in the database. Each point in this TSV space has a set of associated parameter such as its TSV norm and, more importantly, the ID of the DAG they belong to. A DAG ID is simply the index of the DAGINFO structure (see the DAGDBFile below) that contains information about the DAG such as its offset in the DAGDBFile. With this information we can go from the index database to the object database.

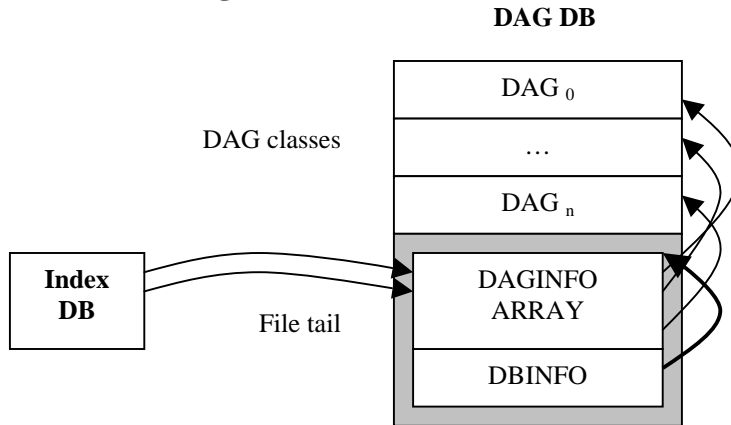
## **DAGDBFile class**

This class extends the functionality of fstream to facilitate dealing with a great number of DAG in a file. The DAGs are stored sequentially from the beginning of file (offset 0). This data is followed by a file tail, which contains information about every DAG in the file as well as some other global parameters.

The DAGs stored in the file will be DAG-derived objects, such as ShockGraph, GestureGraph, and so on. The information in the file tail specifies the DAG class for every DAG in the file so that the correct object type can be created to read the DAG from the file.

All the information in the file tail can be regenerated in case of file corruption with the data stored in each DAG object. Thus, if the file tail isn't present in the database, all the stored DAG will be read sequentially to regenerate the file tail information. This feature allows us to interrupt the construction of the database at any time (for example by ctrl-c, seg fault, etc) without needing to map Unix signals to catch the SIGKILL signal and then save the file tail (which is another valid solution to the problem.)

## File structure diagram



## Individual components of the file

DAG Class
"ClassName" MaxBranFact TotalTSVNorm ...

DAGINFO Class
DAG Type Offset in DB

DBINFO Class
File Tail offset Max TSV dimension

## DAGDBFile class public functions

```
DAGPtr ReadDAG(int nDagId);
```

Reads the offset and DAG type from the cell "nDagId" in the DAGINFO array of the file tail, which is in memory. Then, it creates an object of the corresponding type and reads the DAG at the specified offset. Returns a pointer to the new object.

```
DAGPtr ReadDAG(streamoff nDagOffset, const String& strClassName);
```

Creates an object of type strClassName and reads the DAG at the specified offset. Returns a pointer to the new object.

```
bool AddDAG(const DAG* pDag, bool bReadSaveTailInfo = false);
```

Appends a DAG to the file and updates the tail info in memory. If bReadSaveTailInfo is set to true, the file tail is also saved back to the file. Otherwise, the file tail will be saved when the file is closed.