

Pygr is a long-term open source software project to develop graph database interfaces for the popular Python language, with a strong emphasis on bioinformatics applications ranging from genome-wide analysis of alternative splicing patterns, to comparative genomics queries of multi-genome alignment data. In this demo, we will present tutorials on the basics of how to use Pygr to view any Python data as a “graph database”; write simple but powerful graph queries; use Pygr as a greatly facilitating interface for developing visualization tools (we will use alternative splicing graphs as a tutorial example); provide both container and graph interfaces to existing data in relational databases such as MySQL and Postgres; store and query multiple sequence alignments in a form that can scale to alignments of whole genomes; provide a simple but powerful interface to large sequence databases and automation of tools like BLAST; store and query multi-genome alignments such as the UCSC MAF alignments of many animal genomes.

The basic idea of Pygr is that all Python data can be viewed as a graph whose nodes are objects and whose edges are object relations (in Python, references from one object to another). This has a number of advantages. 1) All data in a Python program become a “database” that can be queried through simple but general graph query tools. In many cases the need to write new code for some task can be replaced by a “database query”. 2) Graph databases are more general and flexible in terms of what they can represent and query than relational databases, which is very important for complex bioinformatics data. 3) Indeed, in Pygr, a “query” is itself just a graph that can be stored and queried in a database, opening paths to automated query construction. 4) Pygr graphs are fully indexed, making queries about edge relationships (which are often unacceptably slow in relational databases) fast. 5) The interface can be very simple and pythonic (it’s just a Mapping).

Examples of the Pygr syntax:

```
graph += node1 # ADD node1 TO graph
graph[node1] += node2 # ADD AN EDGE FROM node1 TO node2
graph[node1][node2]=edge_info # ADD AN EDGE WITH ASSOCIATED edge_info
setschema(node,attr,graph) # ADD node TO graph, WITH graph[node] BOUND AS node.attr
[dict(m) for m in GraphQuery(graph,{1:{2:None,3:None},2:{3:DD(filter=lambda
edge:edge.type=='U11/U12')},3:{})]] # SEARCH graph FOR SUBGRAPH {1->2; 1->3; 2->3}, I.E.
EXONSKIP, WHERE THE SPLICE FROM 2 -> 3 HAS ATTRIBUTE type 'U11/U12'
```

We will illustrate the utility of these capabilities on several genome database query problems. As a tutorial on writing graph queries, we will demonstrate how easy it is to query and categorize complex structures of alternative splicing across a genome-wide database, or develop visualization tools that use the graph as a transparent interface to a genome database in MySQL. Pygr provides gratifyingly simple interfaces to relational databases as containers and graphs, which we will demonstrate extensively in the tutorial.

A second major area in Pygr is representation and query of sequence, annotation, and multiple sequence alignment databases in a way that is scalable to whole genomes. We have previously showed (in our work on Partial Order Alignment) that graphs provide both a compact and algorithmically powerful way to store alignments. Combining this with “interval alignment” makes it scalable and gives a simple interface:

```
hg17=BlastDB('/data/ucsc/hg17') # GET CONTAINER FOR HUMAN GENOME DATABASE
bc12m=hg17['chr22'][16544303:16588541] # GET INTERVAL WITH BCL2L13 GENE
al=hg17.megablast(mouse_bc12,maxseq=1) # GET REPEAT-MASKED MEGABLAST ALIGNMENT, ONLY TOP HIT
al[bc12m[1000:1100]]+=mrna[210:310] #ADD ALIGNMENT OF A 100nt SEGMENT TO mrna SEGMENT
al.storeSQL('test.table',db_cursor) # STORE COMPLETE ALIGNMENT IN RELATIONAL DATABASE
for e in MAFStoredPathMapping(bc12m,'ucsc_maf8',u).edges(): #GET ITS MULTIGENOME ALIGNMENTS
    print str(e.srcPath),str(e.destPath) # PRINT THE ACTUAL ALIGNED SEQUENCE INTERVALS
```

In Pygr, alignments are just another kind of graph, whose nodes are sequence intervals, and edges are alignment relations. This provides a general-purpose facility for working with sets of sequence intervals, sequence annotation databases, and multiple sequence alignments, all queryable via Pygr graph queries. We have implemented different container subclasses to work with these data in memory or to work transparently with data stored in relational databases. The consistency and simplicity of the Pygr framework makes it a good interface both to run external tools like BLAST, and to store or query the results in persistent storage like a MySQL database. We will demonstrate examples in the tutorial.

To illustrate Pygr’s scalability, we will demo using it to query the UCSC BLASTZ multiple alignment of 8 genomes (generated by David Haussler’s group). Pygr can be used as a front-end to a large set of back-end databases (e.g. the 8 genomes, and the very large multiple alignment), only loading the alignment regions you are actually working with, storing them compactly as interval alignments, only getting sequence when it’s actually needed, and then doing so with fast tools that just “slice” out the desired segment. This is done transparently, without the user having to worry about such optimizations, by taking advantage of standard Python idioms, e.g. to actually get the sequence for an interval obj, just use `str(obj)`.

Pygr is an open source project, available as prerelease software (v0.1) under the GPL from <http://www.bioinformatics.ucla.edu/leelab/db>. We hope these tools will be helpful to other developers, and are eager for other developers to participate in the ongoing development of Pygr and its applications. This work has been supported by the NSF ITR, and by the NIH Center for Computational Biology at UCLA.