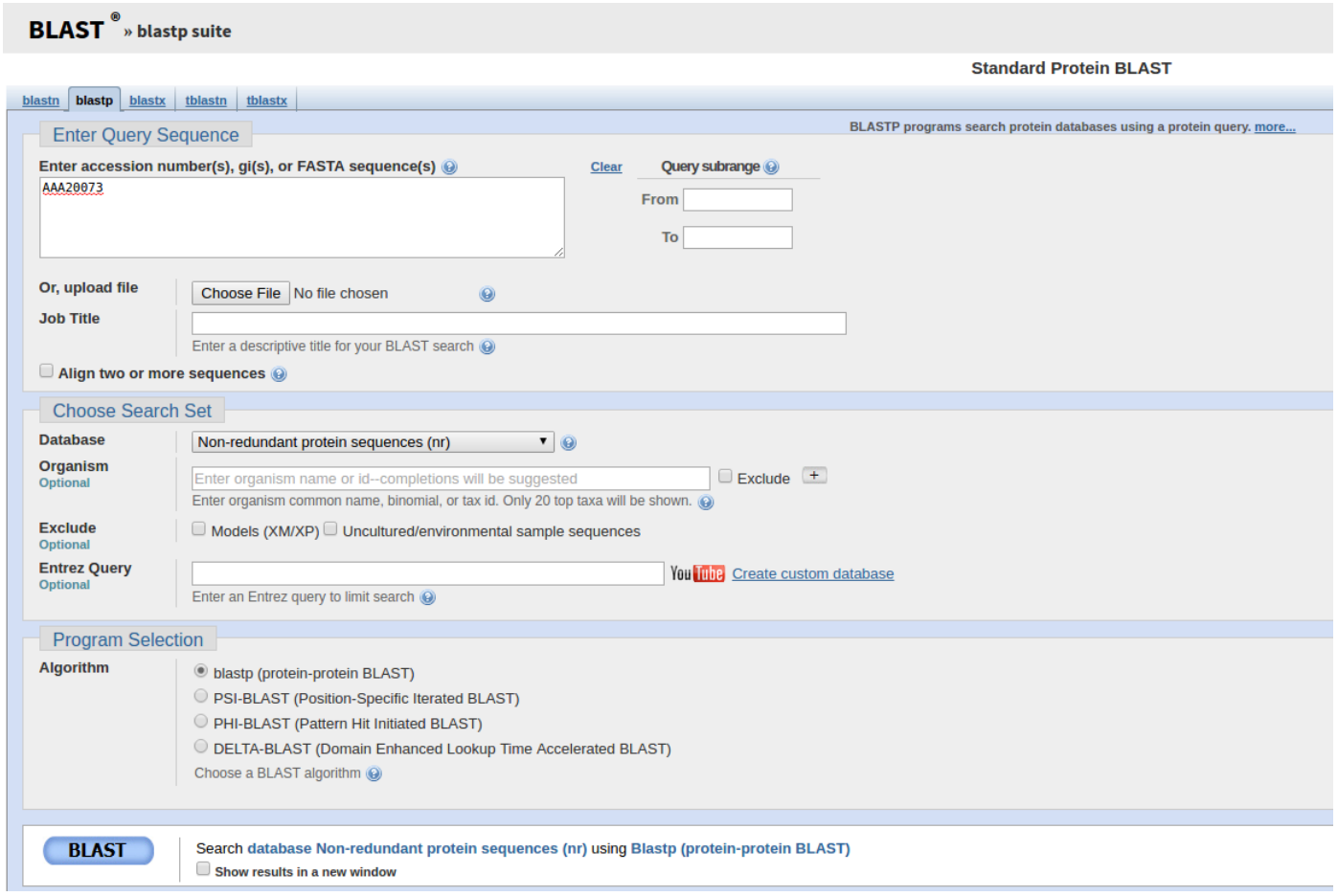
BLAST analysis at Genbank

I am interested in the ACC1 gene (http://sgd-beta.stanford.edu/locus/S000005299/overview) of *S. cerevisiae*.

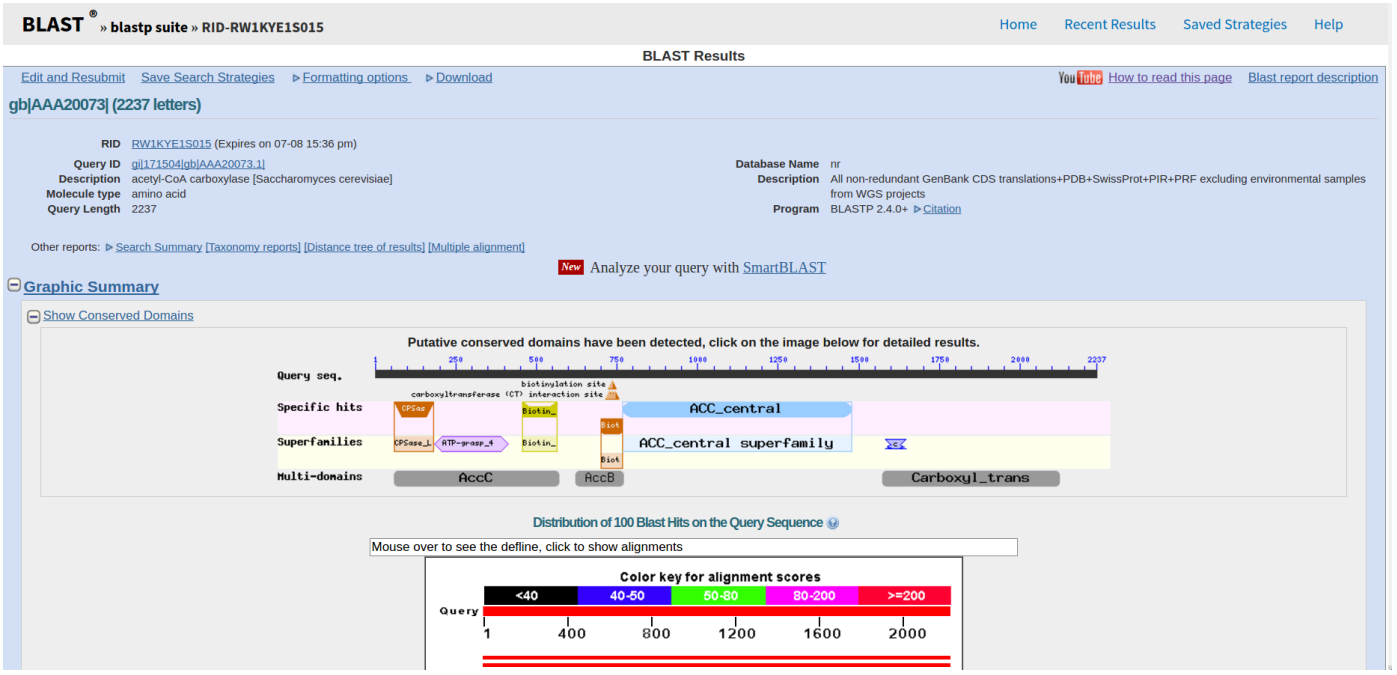
I would like to have the protein sequences of the 100 best BLAST hits from Genbank using the ACC1p protein sequence as query.

The accession number of the ACC1 protein sequence is AAA20073 (http://www.ncbi.nlm.nih.gov/protein/171504/).

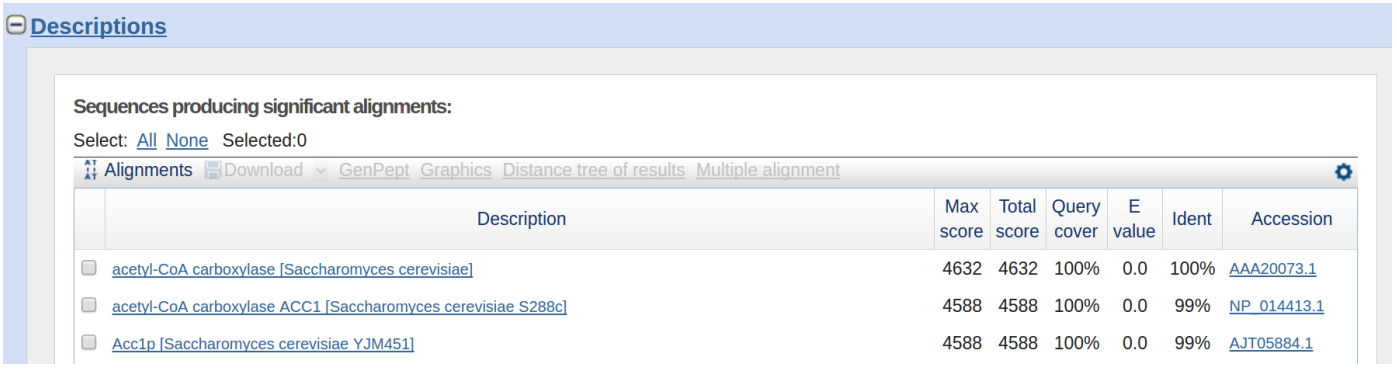
This CAN be done manually here (https://blast.ncbi.nlm.nih.gov/Blast.cgi?PROGRAM=blastp&PAGE_TYPE=BlastSearch&LINK_LOC=blasthome&QUERY=AAA20073) like this:



When the blast search is finished, you will be taken to a results page that looks like this:



If you scroll down the page, you can click on the links on the right side of the page for each result. These links will give you the Genbank file for each of the sequences that produced the blast hists with your sequence like this:



However, it is very laborious. You will easily introduce errors. What if you want to redo the analysis frequently to see if new relevant sequences have been found? What if your boss (boss.jpg) tells you he needs the same results for another protein? or the thousand best blast hits?

Biopython

from Wikipedia (https://en.wikipedia.org/wiki/Biopython)

"The Biopython Project is an open-source collection of non-commercial Python tools for computational biology and bioinformatics, created by an international association of developers.

It contains classes to represent biological sequences and sequence annotations, and it is able to read and write to a variety of file formats.

It also allows for a **programmatic means of accessing online databases of biological information**, such as those at NCBI. Separate modules extend Biopython's capabilities to sequence alignment, protein structure, population genetics, phylogenetics, sequence motifs, and machine learning.

Sounds good!

Lets look in more detail. There is an online tutorial for Biopython and it contains information on what we want here (http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc86).

We can use the qblast() function provided by Biopython as in the tutorial. Qblast produces results in XML (https://en.wikipedia.org/wiki/XML) format.

This analysis takes some time, so we would like to save the results in a file just as they did in the tutorial.

We call this file "my_blast.xml" in the code cell below:

In [20]:

```
from Bio.Blast import NCBIWWW

result_handle = NCBIWWW.qblast("blastp",
                               "nr",
                               "AAA20073",
                               hitlist_size=100,
                               ncbi_gi=True,
                               alignments = 1,
                               expect=10.0)

with open("my_blast.xml", "w") as f:
    f.write(result_handle.read())

result_handle.close()
```

The my_blast.xml (my_blast.xml) is so easy to understand by looking at it. It does contain information on every High-scoring Segment Pair HSP (https://en.wikipedia.org/wiki/BLAST#Algorithm) for each one of the one hundred best hits and the parameters that were used in the analysis.

Biopython has a parser (http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc91) that can be used to extract ACCESSION (https://en.wikipedia.org/wiki/Accession_number_%28bioinformatics%29) numbers for all the results:

In [27]:

```
from Bio.Blast import NCBIXML

result_handle = open("my_blast.xml")

blast_record = NCBIXML.read(result_handle)

ids = []

for h in blast_record.alignments:
    ids.append(h.accession)

result_handle.close()
```

we have one hundred ACCESSION numbers in the ids variable.

In [28]:

```
len(ids)
```

Out[28]:

```
100
```

The first five numbers:

In [29]:

```
ids[:5]
```

Out[29]:

```
[u'AAA20073', u'NP_014413', u'AJT05884', u'EDN62822', u'AJT27522']
```

Now we would like to get the sequences for all of the ACCESSION numbers. This is trickier than we might expect, due to the design of the database.

We have to make a string containing all of the numbers divided by a blank:

In [30]:

```
accs=ids

query  = " ".join(accs)
```

We have to set a variable to say how many results we want. We call this variable retmax.

In [31]:

```
retmax=100
```

We will use functionality of the Entrez module. We have to tell Genbank who we are when we use their service. We can do this by setting the Entrez.email variable.

In [32]:

```
from Bio import Entrez

Entrez.email = "bjornjobb@gmail.com"
```

First we need the gi number associated with each accession number.

We will use the biopython wrapper for the Entrez E-utilities server programs. Here is a tutorial from NCBI (http://www.ncbi.nlm.nih.gov/books/NBK25501/).

Entrez.esearch (http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc112) can be used to search the E-utilities programs.

In [33]:

```
handle = Entrez.esearch( db="protein", term=query, retmax=retmax )

giList = Entrez.read(handle)['IdList']
```

In [35]:

```
giList[:5]
```

Out[35]:

```
['171504', '6324343', '767226550', '151944544', '767249005']
```

The code below posts a search on Entrez that we can fetch later

In [40]:

```
handle = Entrez.epost(db="protein", id=",".join(giList), rettype="fasta", retmode="

result = Entrez.read(handle)

search_results = result

webenv, query_key  = search_results["WebEnv"], search_results["QueryKey"]
```

Below, we download the results in batches of 100 sequences (1 batch in this case).

In [41]:

```
db="protein"

batchSize = 100

from Bio import SeqIO

for start in range( 0, len(giList), batchSize ):

    handle = Entrez.efetch(db=db,
                           rettype="gb",
                           retstart=start,
                           retmax=batchSize,
                           webenv=webenv,
                           query_key=query_key)

    with open("batch{}.gb".format(start+1), "w") as f:
        f.write(handle.read())
```

File with all the results:

Genbank files (batch1.gb)