



Example with GCOptimization

Kevin Keraudren
Imperial College London

October 31st, 2013

Cython overview

- Syntax between Python and C (keyword `cdef`)
- C/C++ code automatically generated, then compiled into a Python module
- For a speed gain, variables must be declared
- C++ templates must be instantiated (compiled code)
- Choose between accessing low-level C++ or a blackbox

Documentation: `docs.cython.org`

Learn from examples:

`scikit-learn`, `scikit-image`, `github.com/amueller`

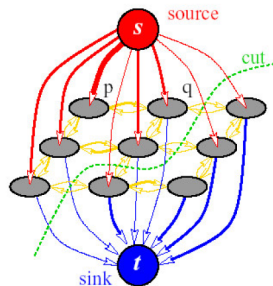
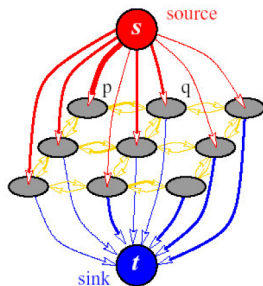
How to?

- 1 Organize your C/C++ code
- 2 Write `module.pyx`
- 3 Write `setup.py`
- 4 Build

Example 1

Interface the whole API

Graphcut (Boykov & Kolmogorov)



$$V(p, q) = \begin{cases} \frac{1}{\|p - q\|_2} e^{-(l_p - l_q)^2 / 2\sigma^2} & \text{if } l_p \geq l_q \\ \frac{1}{\|p - q\|_2} & \text{if } l_p < l_q \end{cases}$$

σ : noise estimate

GCoptimisation (Boykov & Kolmogorov)

```
template <typename captype,  
         typename tcaptype,  
         typename flowtype> class Graph {  
public:  
    ...  
    Graph( int node_num_max, int edge_num_max,  
           void (*err_function)(const char *) = NULL);  
    void add_edge( node_id i, node_id j,  
                  captype cap, captype rev_cap);  
    void add_tweights( node_id i,  
                      tcaptype cap_source, tcaptype cap_sink);  
    flowtype maxflow( bool reuse_trees = false,  
                    Block<node_id>* changed_list = NULL);  
    termtype what_segment( node_id i,  
                          termtype default_seg = SOURCE);  
    ...  
}
```

Begin your module.pyx

```
import numpy as np
```

```
cimport numpy as np
```

```
np.import_array()
```

```
ctypedef double catype
```

```
ctypedef double tcatype
```

```
ctypedef double flowtype
```

Declare what you need from C++

```
cdef extern from "graph.h":  
    cdef cppclass Graph[captype, tcaptype, flowtype]:  
        Graph( size_t, size_t )  
        size_t add_node(size_t)  
        void add_edge(size_t, size_t, captype, captype)  
        void add_tweights(size_t, tcaptype, tcaptype)  
        flowtype maxflow()  
        int what_segment(size_t)
```


Create your Python class

```
cdef class PyGraph:
    # hold a C++ instance which we're wrapping
    cdef Graph[capttype,tcapttype,flowtype] *thisptr
    def __cinit__(self, size_t nb_nodes, size_t nb_edges):
        self.thisptr = new Graph[capttype,
                                   tcapttype, flowtype](nb_nodes,nb_edges)
    def __dealloc__(self):
        del self.thisptr
```

Create your Python class

```
def add_node(self, size_t nb_nodes=1):  
    self.thisptr.add_node(nb_nodes)  
def add_edge(self, size_t i, size_t j,  
              captype cap, captype rev_cap):  
    self.thisptr.add_edge(i,j,cap,rev_cap)  
def add_tweights(self, size_t i,  
                  tcaptype cap_source, tcaptype cap_sink):  
    self.thisptr.add_tweights(i,cap_source,cap_sink)  
def maxflow(self):  
    return self.thisptr.maxflow()  
def what_segment(self, size_t i):  
    return self.thisptr.what_segment(i)
```

Write setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
from numpy.distutils.misc_util import get_numpy_include_dirs

setup(
    cmdclass = {'build_ext': build_ext},
    ext_modules = [
        Extension( "graphcut",
            [ "graphcut.pyx",
              "../maxflow-v3.02.src/graph.cpp",
              "../maxflow-v3.02.src/maxflow.cpp" ],
            language="c++",
            include_dirs=get_numpy_include_dirs()+["../maxflow-v3.02.src"],
        )
    ]
)
```

And build:

```
python setup.py build_ext --build-temp tmp \\  
--build-lib lib \\  
--pyrex-c-in-temp
```

And use it!

```
from lib import graphcut
G = graphcut.PyGraph(nb_pixels,nb_pixels*(8+2))
G.add_node(nb_pixels)
...
print "building graph..."
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        for a,b in neighbourhood:
            if ( 0 <= i+a < img.shape[0]
                and 0 <= j+b < img.shape[1] ):
                dist = np.sqrt( a**2 + b**2 )
                if img[i,j] < img[i+a,j+b]:
                    w = 1.0/dist
                else:
                    w = np.exp(-(img[i,j] - img[i+a,j+b])**2
                                w /= 2.0 * std**2 * dist
                G.add_edge( index(i,j,img),
                           index(i+a,j+b,img),
                           w, 0 )
```

Result



Example 2

Use C++ as a blackbox

Declare C++ function

```
cdef extern from "_graphcut.h":  
    void _graphcut( voxel_t*,  
                    int, int,  
                    double,  
                    unsigned char*,  
                    unsigned char* )
```

And use it!

```
def graphcut( np.ndarray[voxel_t, ndim=2, mode="c"] img,
              np.ndarray[unsigned char, ndim=2, mode="c"] mask,
              double std ):

    cdef np.ndarray[unsigned char,
                    ndim=2,
                    mode="c"] seg = np.zeros( (img.shape[0],
                                                img.shape[1]),
                                                dtype='uint8')

    print "starting graphcut..."
    _graphcut( <voxel_t*> img.data,
               img.shape[0], img.shape[1],
               std,
               <unsigned char*> mask.data,
               <unsigned char*> seg.data )

    return seg
```


Result



Timing

Example 1: 18.01s

Example 2: 0.37s

Nearly 50 times faster...

Another result...



Conclusion

- Huge speedup for a low amount of code
- Perfect if C++ code already exists
- Make sure your Python code is optimised (good use of `numpy`) before using `cython`

Examples are available for download

Thanks!