week06-02–min-cut-notes

February 20, 2025

Write `V0` for the interior vertices of our directed graph, so that the vertices `V` of `G` are the union of `[s,t]` and `V0`

We want to describe all possible partitions of `V` into an `s-group` and a `t-group`.

Our starting point is a library function that produces all of the sub-lists of a given list.

That library function comes from the `itertools` library and is named `combinations`.

[12]:
```python
from itertools import combinations, chain

list(combinations([1,2,3,4],2))
```

[12]:
```
[(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]
```

Now we need a way to concatenate the output of `combinations(ll,j)` for varying values of `j`.

[73]:
```python
def powerset(ll):
    lol = [ combinations(ll,j) for j in range(len(ll) + 1) ]
    return list(chain.from_iterable(lol))

powerset([1,2,3,4])
```

[73]:
```
[(),
 (1,),
 (2,),
 (3,),
 (4,),
 (1, 2),
 (1, 3),
 (1, 4),
 (2, 3),
 (2, 4),
 (3, 4),
 (1, 2, 3),
 (1, 2, 4),
 (1, 3, 4),
 (2, 3, 4),
 (1, 2, 3, 4)]
```

Let's suppose we have represented the (weighted) edges in our graph `G` using a dictionary `edges`. The keys to the dictionary are pairs `(v,w)` where `v` and `w` are vertices. and the value `edges[(v,w)]` is the `capacity` of the edge.

So we could have something like the following:

[20]:
```
verts0 = ['a','b','c','d' ]
verts = ['s','t'] + verts0
edges = { ('a','b'):  10,
          ('a','c'):  20,
          ('b','d'):  30,
          ('c','d'):  40,
#
          ('s','a'):  5,
          ('s','d'):  7,
          ('b','t'):  8,
          ('c','t'): 10
        }
```

Now, given an `s-group` `I`, we want to compute the corresponding `cut-value` based on our capacities. So we have to compute the sum of the `capacity` for each edge in `edges` which connects a vertex in the `s-group` to the `t-group` `V - I`

Well, we can sum the values in a list using the `sum` function:

[23]:
```
sum([1,2,3])
```

[23]:
```
6
```

[24]:
```
sum(range(50))
```

[24]:
```
1225
```

[55]:
```
def isCutEdge(I,v,w):
    # I: the s-group
    # returns True is the edge v->w goes from the s-group to the t-group
    # and False otherwise
    if v in I and (not (w in I)):
        return True
    else:
        return False

def cutValueForGroup(edges,I):

    # determine the cut-value determined by s-group I

    cut_edges = [ edges[(v,w)] for (v,w) in edges.keys() if isCutEdge(I,v,w) ]

    return sum(cut_edges)
```

[74]:

```
cutValue(edges,['s'])
```

[74]:

```
12
```

[75]:

```
def minCut(verts0,edges):
    ps = powerset(verts0)
    cuts = [ cutValueForGroup(edges, ('s',) + I) for I in ps ]
    return min(cuts)
```

[70]:

```
verts
```

[70]:

```
['s', 't', 'a', 'b', 'c', 'd']
```

[71]:

```
edges
```

[71]:

```
{('a', 'b'): 10,
 ('a', 'c'): 20,
 ('b', 'd'): 30,
 ('c', 'd'): 40,
 ('s', 'a'): 5,
 ('s', 'd'): 7,
 ('b', 't'): 8,
 ('c', 't'): 10}
```

[76]:

```
minCut(verts0,edges)
```

[76]:

```
5
```

[ ]: