A semiring for computing all paths and cycles in a graph

Vladimir Batagelj
IMFM - Institute of Mathematics, Physics and Mechanics,
Jadranska 19, 1 000 Ljubljana, Slovenia
vladimir.batagelj@fmf.uni-lj.si

University of Primorska, Andrej Marušič Institute, Koper

July 21, 2017

1 Semiring

In a graph $G = (V, L)$, a *path* is a walk with all its nodes different, and a *cycle* is a closed walk with all its internal nodes different. With V^* we denote the set of all sets of nonempty sequences (descriptions of simple walks, i.e. paths and cycles) over set of nodes V.

We construct a semiring $(V^+, \cup, \cdot, \emptyset, E)$ as follows. Let $A, B \in V^+ \setminus \emptyset$ then
$A \cdot B = \{a \bullet b : a \in A, b \in B\}$ where

$$a \bullet b = \begin{cases} & \text{last}(a) = \text{first}(b) \wedge \\ a \circ b & ((\text{first}(a) = \text{last}(b) \wedge \text{set}(\text{bf}(a)) \cap \text{set}(\text{bf}(b)) = \emptyset) \vee \\ & (\text{first}(a) \neq \text{last}(b) \wedge \text{set}(a) \cap \text{set}(\text{bf}(b)) = \emptyset)) \\ \text{nothing} & \text{otherwise} \end{cases}$$

$\circ$ is the operation of *concatenation* of paths and $\text{bf}(a)$ is the operation butfirst – returns a sequence a with the first item removed. We additionally set

$$\emptyset \cdot A = A \cdot \emptyset = \emptyset$$

The nevtral element for semiring multiplication is $E = \{(v) : v \in V\}$.

Assuming that the set of nodes V is finite then also the V^+ is finite. Because the semiring is idempotent on a finite set it is also complete – there exists a closure A^* for each $A \in V^+$.

An element $C \in V^+$ is *cyclic* iff for all $c \in C$ it holds $\text{first}(c) = \text{last}(c)$. It is easy to verify that for every cyclic element C it holds

$$C^\star = E \cup C$$

Kleene, Warshall, Floyd and Roy are contributed to the development of the procedure which final form was given by Fletcher.

$\mathbf{C}_0 := \mathbf{W}$;
for $k := 1$ **to** n **do begin**
  for $i := 1$ **to** n **do for** $j := 1$ **to** n **do**
    $c_k[i,j] := c_{k-1}[i,j] + c_{k-1}[i,k] \cdot (c_{k-1}[k,k])^\star \cdot c_{k-1}[k,j]$;
  $c_k[k,k] := 1 + c_k[k,k]$;
end;
$\mathbf{W}^\star := \mathbf{C}_n$;

If we delete the statement $c_k[k,k] := 1 + c_k[k,k]$ we obtain the algorithm for computing the strict closure $\overline{\mathbf{W}}$.

For our semiring it holds

$$c_{k-1}[i,k] \cdot (c_{k-1}[k,k])^\star \cdot c_{k-1}[k,j] = c_{k-1}[i,k] \cdot c_{k-1}[k,j]$$

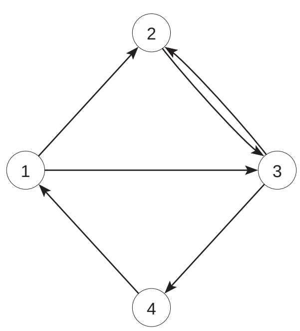
Since the semiring is idempotent the Fletcher's algorithm can be performed in place – we can omit indices in c_k. The algorithm for a strict closure gets the form:

$\mathbf{C} := \mathbf{W}$;
for $k := 1$ **to** n **do for** $i := 1$ **to** n **do for** $j := 1$ **to** n **do**
  $c[i,j] := c[i,j] \cup c[i,k] \cdot c[k,j]$;
$\overline{\mathbf{W}} := \mathbf{C}$;

2 Python

```
# Computing a matrix C of all paths in a given graph G
# V.B.  13.5.2012 / 14.7.2017 / 21.7.2017

G = [ [ 0, 1, 1, 0],
      [ 0, 0, 1, 0],
      [ 0, 1, 0, 1],
      [ 1, 0, 0, 0] ]

n = len(G)
E = set([(v+1,) for v in range(n)])

def transit(G):
  R = G; n = len(G)
  for u in range(n):
    for v in range(n):
      C = set()
      if G[u][v] != 0: C.add((u+1,v+1))
      R[u][v] = C
  return R

def times(A,B):
  C = set()
  if (not A)|(not B): return C
  for a in A:
    for b in B:
      if a[-1] == b[0]:
        if a[0] == b[-1]:
          if set(a[1:]).isdisjoint(set(b[1:])): C.add(a+b[1:])
        else:
          if set(a).isdisjoint(set(b[1:])): C.add(a+b[1:])
  return C

def closure(R): # Fletcher's algorithm / strict closure
  n = len(R); C = R
  for k in range(n):
    for u in range(n):
```

```
      for v in range(n):
        C[u][v] = C[u][v] | times(C[u][k],C[k][v])
  return C

def output(R):
  n = len(R)
  for u in range(n):
    for v in range(n): print(u+1,v+1,R[u][v])

def outvec(D):
  for u in range(len(D)): print(u+1,D[u])

def hamilton(D):
  n = len(D); H = [0]*n
  for u in range(n):
    S = set()
    for p in D[u][u]:
      if len(p)>n: S.add(p)
    H[u] = S
  return H

print('matrix G'); output(G)

R = transit(G)
print('\ntransition matrix R'); output(R)

C = closure(R)
print('\nclosure matrix C'); output(C)

H = hamilton(C)
print('\nhamilton cycles vector H'); outvec(H)
```



3 Results

```
>>>
====== RESTART: C:/Users/batagelj/work/Python/WoS/Preprint/closureS.py ======
matrix G
1 1 0
1 2 1
1 3 1
1 4 0
2 1 0
2 2 0
2 3 1
2 4 0
3 1 0
3 2 1
3 3 0
3 4 1
4 1 1
4 2 0
```

```
4 3 0
4 4 0

transition matrix R
1 1 set()
1 2 {(1, 2)}
1 3 {(1, 3)}
1 4 set()
2 1 set()
2 2 set()
2 3 {(2, 3)}
2 4 set()
3 1 set()
3 2 {(3, 2)}
3 3 set()
3 4 {(3, 4)}
4 1 {(4, 1)}
4 2 set()
4 3 set()
4 4 set()

closure matrix C
1 1 {(1, 3, 4, 1), (1, 2, 3, 4, 1)}
1 2 {(1, 2), (1, 3, 2)}
1 3 {(1, 3), (1, 2, 3)}
1 4 {(1, 2, 3, 4), (1, 3, 4)}
2 1 {(2, 3, 4, 1)}
2 2 {(2, 3, 4, 1, 2), (2, 3, 2)}
2 3 {(2, 3)}
2 4 {(2, 3, 4)}
3 1 {(3, 2, 3, 4, 1), (3, 4, 1)}
3 2 {(3, 2), (3, 4, 1, 2)}
3 3 {(3, 2, 3), (3, 4, 1, 2, 3), (3, 4, 1, 3)}
3 4 {(3, 4), (3, 2, 3, 4)}
4 1 {(4, 1)}
4 2 {(4, 1, 2), (4, 1, 3, 2)}
4 3 {(4, 1, 2, 3), (4, 1, 3)}
4 4 {(4, 1, 2, 3, 4), (4, 1, 3, 4)}

hamilton cycles vector H
1 {(1, 2, 3, 4, 1)}
2 {(2, 3, 4, 1, 2)}
3 {(3, 4, 1, 2, 3)}
4 {(4, 1, 2, 3, 4)}
>>>
```

References

[1] Batagelj, V: Semirings for social networks analysis. J Math Sociol 19 (1): 53-68 1994.

[2] Batagelj,V.: Efficient Algorithms for Citation Network Analysis. arXiv:cs/0309023