SickPropagation

Filippo Magi

February 17, 2022

1 Propagazione della malattia

Algorithm 1 Ricerca del massimo flusso con propagazione della malattia

Require: Rete (G, u, s, t)
Ensure: valore del flusso massimo
  $vuoti \leftarrow$ pila vuota di nodi
  while TRUE **do**
    $f \leftarrow$ DoBfs$(G, vuoti)$
    if $f = 0$ **then**
      break
    end if
    $mom \leftarrow t$
    while $mom \neq s$ **do**
      m.PreviousEdge.AddFlow(f)
      if $u_f(mom$.PreviousEdge$) = 0$ **then**
        $vuoti.push(mom)$
      end if
      $mom \leftarrow mom$.PreviousNode
    end while
  end while
  return flusso uscente da s

Algorithm 2 DoBfs con propagazione della malattia

Require: rete (G, u, s, t), pila di nodi *noCaps*
Ensure: flusso inviato al nodo t

```
coda ← coda vuota di nodi
malati ← coda vuota di nodi
fromSource ← false
if noCaps.isEmpty then
  coda.enqueue(s)
  fromSource ← true
else
  repaired ← true
  while ¬noCaps.isEmpty do
    noCap ← noCaps.pop()
    Repair(noCap) {controllo se c'è un nodo con label = noCap.label-1 e con
capacità o flusso diversa da 0, aggiornando i dati}
    if noCap non è stato riparato then
      malati.enqueue(noCap)
      repaired ← false
    end if
  end while
  if repaired then
    f ← GetFlow(t) {recupero il flusso inviabile nel percorso descritto tra s
e t }
    return f
  end if
  while ¬malati.isEmpty do
    noCap ← malati.Dequeue()
    v ← SickPropagation(G, noCap, coda)
    if v ≠ 0 then
      return v
    end if
    if prima iterazione del ciclo then
      x ← noCap
    end if
  end while
  if t.valid then
    GetFlow(t)
  end if
  for all n ∈ N(G)|n.label ≥ x.label do
    n.Reset()
  end for
  if coda.isEmpty then
    coda ← n ∈ G|n.label = (x.label − 1)
  end if
end if
```

```
while ¬coda.isEmpty do
  element ← coda.Dequeue()
  for all edge ← element.Edges do
    n ← edge.nextNode
    p ← edge.previousNode
    if (n.visited ∧ p.visited) ∨ ¬element.valid then {se il nodo è invalido è
considerato come non visitato}
      continue
    end if
    if p = element ∧ u_f(edge) > 0 ∧ (n.label ≥ p.label ∨ ¬n.valid ∨
fromSource) then
      n.update(p, edge) {(label,visitato, nodo precedente, nel caso riparo il
nodo)}
      if n = t then
        return GetFlow(n)
      else
        coda.enqueue(n)
      end if
    else if n = element ∧ f(edge) > 0 ∧ (p.label ≥ n.label ∨ fromSource ∨
¬p.valid) then
      p.update(n,edge)
      if p = t then
        return GetFlow(p)
      else
        coda.enqueue(p)
      end if
    end if
  end for
end while
return 0
```

Algorithm 3 SickPropagation

Require: rete (G, u, s, t), Nodo*node*, coda di nodi *coda*
Ensure: possibile flusso inviato verso t (partendo da n,conoscendo i valori antecedenti a n), 0 altrimenti
  malati $\leftarrow$ coda vuota di nodi
  malati.Enqueue(*node*);
  while $\neg malati.isEmpty$ **do**
    $m \leftarrow malati$.Dequeue()
    Repair(m)
    if m non è stato riparato **then**
      malati.enqueue($n \in A(m) | n$.previousNode$= m$) {con $A(m)$ si intendono i nodi adiacenti di m}
    else if $m = t$ **then**
      return GetFlow(m)
    else
      coda.enqueue(m)
    end if
  end while
  return 0