A Distributed Algorithm for the Maximum Flow Problem

Thuy Lien PHAM, Ivan LAVALLÉE, Marc BUI, Si Hoang DO

LRIA, Université Paris 8, France

ISPDC
July 5, 2005
Outline

Introduction
  Maximum Flow Problem
  Sequential Goldberg-Tarjan’s preflow-push algorithm

Distributed Computing
  Principle
  Illustrations
  Adaptative algorithm
  Complexity
Outline

Introduction
  Maximum Flow Problem
  Sequential Goldberg-Tarjan’s preflow-push algorithm

Distributed Computing
  Principle
  Illustrations
  Adaptative algorithm
  Complexity
Notations & Definitions

Network

• connected directed graph $G = (V, E); |V| = n$ and $|E| = m$
• source $s$, sink $t$
• capacity function $c(e) \in \mathbb{R}^+, \forall e \in E$

Flow

• a function $f : E \rightarrow \mathbb{R}^{+*}$:
  1. $0 < f(e) \leq \text{cap}(e), \forall e \in E$ - capacity constraints
  2. $\sum_{(u, v) \in E} f(u, v) = \sum_{(v, w) \in E} f(v, w), \forall v \in V \setminus \{s, t\}$ - flow conservation constraints
Notations & Definitions

- **excess** of a node $v \in V \setminus \{s, t\}$:

  $$excess(v) = \sum_{(u, v) \in E} f(u, v) - \sum_{(v, w) \in E} f(v, w)$$

  $$\rightarrow excess(v) = 0, \forall v \in V \setminus \{s, t\}$$

- **flow value** $f$ is defined:

  $$f = excess(t)(= -excess(s))$$

- **Max flow problem**: find flow from $s$ to $t$ with maximal value
Residual network

- let $f$ be a flow in $G = (V, E)$
- the residual network induced by $f$: $G_f = (V, E_f)$
- Example
Existing Algorithms & Complexities

Augmenting path algorithms

- sequential: $O(n^3)$ time
- distributed:
  - synchronous: $O(n^3)$ msg complexity and $O(n^2)$ time complexity
  - asynchronous: $O(kn^3)$ msg complexity and $O(n^2 \log n)$ time complexity, $2<k<n$

Preflow-push algorithms

- sequential: best implementation $O(n^3)$
- distributed:
  - asynchronous: this paper $O(n^2 m)$ msg complexity and $O(n^2)$ time complexity
Sequential preflow-push algorithm

- Algorithm maintains a preflow: some node don’t hold flow conservation constraint at intermediate step, i.e. \( \text{excess}(v) \geq 0 \) (we call active node if excess > 0)
- Nodes have height
- Flow is pushed to lower node
- Nodes sometimes are lifted
- Source and sink nodes never lift
Sequential preflow-push algorithm

compute exact distances
Sequential preflow-push algorithm
Sequential preflow-push algorithm

saturate outgoing arcs from s
Sequential preflow-push algorithm
Sequential preflow-push algorithm
Sequential preflow-push algorithm

push flow from red to orange node
Sequential preflow-push algorithm

lift red node
Sequential preflow-push algorithm
Sequential preflow-push algorithm

push flow from red to orange node
Sequential preflow-push algorithm

push flow from red to orange node
Sequential preflow-push algorithm
Sequential preflow-push algorithm

push flow from red to orange node
Sequential preflow-push algorithm
Sequential preflow-push algorithm

push flow from red to orange node
Sequential preflow-push algorithm

lift red node
Sequential preflow-push algorithm

push flow from red to orange node
Sequential preflow-push algorithm

push flow from red to orange node
Sequential preflow-push algorithm

push flow from red to orange node
Sequential preflow-push algorithm

push flow from red to orange node
Sequential preflow-push algorithm
Sequential preflow-push algorithm

MaxFlow = 19
Outline

Introduction
- Maximum Flow Problem
- Sequential Goldberg-Tarjan’s preflow-push algorithm

Distributed Computing
- Principle
- Illustrations
- Adaptative algorithm
- Complexity
Model of computation

- Sequential processes at nodes
- Bidirectional communication links
- Each process runs the same local algorithm:
  - Send messages to neighbors
  - Wait for incoming messages and process them
High-level description

1. Initialization of nodes’ heights
   - in Breadth First Search way
   - node’s height is the minimum distance from that node to sink

2. The excess is pushed from the source downhill towards the sink
   - push: from higher node to lower node
   - lift: increase to a threshold in order to push all remaining node excess
   - a push can be NOT successful
High-level description

1. Initialization of nodes’ heights
   - in Breadth First Search way
   - node’s height is the minimum distance from that node to sink

2. The excess is pushed from the source downhill towards the sink
   - push: from higher node to lower node
   - lift: increase to a threshold in order to push all remaining node excess
   - a push can be NOT successful
Variables at node

- node’s type: \( type_v = (\text{SOURCE} \mid \text{SINK} \mid \text{TRANSIT}) \);
- node’s state: \( state_v = (\text{inactive} \mid \text{active}) \);
- node’s excess value of flow: \( excess_v \)
- node’s height: \( height_v \)
- list of neighbors and their current heights: \( neighborlist_v, height_v[u], \forall u \in neighborlist_v \);
- residual capacities of links (residual arcs): \( r_v[u], \forall u \in neighborlist_v : \text{real} \);
- number of init messages (for 1st phase): \( nbInitHeightMsgs \);
Types of messages

- INIT (src, dest, height)
- PUSH-REQUEST (src, dest, flow value to push)
- PUSH-REQUEST-ANS (src, dest, value, NOK)
- NEW-HEIGHT (src, dest, height)
Illustration
Illustration
Illustration
Illustration
Illustration
Illustration
Illustration
Illustration
Illustration
Illustration
Illustration
Illustration
Illustration
Illustration
Illustration
Illustration
Illustration
Illustration
Illustration
Adaptative algorithm - Increasing of an arc
Adaptative algorithm - Increasing of an arc
Adaptative algorithm - Increasing of an arc
Adaptative algorithm - Increasing of an arc
Adaptative algorithm - Increasing of an arc
Adaptative algorithm - Increasing of an arc
Adaptative algorithm - Increasing of an arc
Adaptative algorithm - Increasing of an arc
Adaptative algorithm - Increasing of an arc
Adaptative algorithm - Increasing of an arc
Adaptative algorithm - Increasing of an arc

![Diagram showing an adaptive algorithm with labels for nodes and edges, indicating increasing values of a parameter r for each node and edge.](image-url)
Adaptative algorithm - Increasing of an arc
Adaptative algorithm - Decreasing of an arc
Adaptative algorithm - Decreasing of an arc
Adaptative algorithm - Decreasing of an arc
Adaptative algorithm - Decreasing of an arc

![Diagram of a network with nodes labeled a, b, c, d, s, and t, and edges with labels r=0, r=1, r=2, etc. The diagram illustrates the process of an adaptative algorithm and the decreasing of an arc.]
Adaptative algorithm - Decreasing of an arc
Adaptative algorithm - Decreasing of an arc
Adaptative algorithm - Decreasing of an arc
Adaptative algorithm - Decreasing of an arc
Adaptative algorithm - Decreasing of an arc
Measures of complexities

Communication cost

- Total number of messages sent

Timing cost

- local computation is much faster than inter-processes communication
- measure of computation time = number of rounds of message-passing (pulses)
- during a pulse, nodes can (possibly):
  - receive messages
  - performs local computation
  - send messages which will be received at the beginning of the following pulse
Message Complexity

Number of messages in the first phase

- Breadth first search is $O(mn)$ message complexity.

Number of messages in the second phase
Message Complexity

**Number of lifts**

- for all \( v \): \( \text{height}_v < 2n \)
  - \( \text{height}_s \) remains \( n \)
  - when \( v \) is lifted and pushed, no steed arc is created
  - \( \text{height}_v \) is lower bound on path length from \( v \) to \( s \)

- when each node lifts, its height increases at least 1
  → number of lifts is less than
  \[ (n + 1) + (n + 2) + ... + (2n - 2) = 3n^2/2 - 5n/2 \]

- a push is not successful when the receiver is lifted → number of PUSH-REQUEST-NOK msg is \( O(n^2) \)

- when a node lifts, a NEW-HEIGHT msg is sent across each link to its neighbors → number of NEW-HEIGHT msg across this link is at most \( 4n \), in total over network \( O(mn) \)
Message Complexity

Number of pushes

- Types of pushes
  - saturating: sends $\delta = r(u, v)$ across $(u, v)$
  - non-saturating: sends $\delta < r(u, v)$
  - Number of PUSH-REQUEST msg is total number of pushes

- Number of saturating pushes
  - a saturating push across $(u, v)$ must have $\text{height}_u \geq \text{height}_v + 1$
  - a saturating push across $(v, u)$ must have $\text{height}_v \geq \text{height}_u + 1$
  - $\forall v \in V$ we have $\text{height}_v < 2n \Rightarrow$ number of saturating pushes across original arc $uv < n$
  - $O(mn)$ saturating pushes
Message Complexity

Non-saturating pushes

- potential function $\phi = \sum_{\text{excess}(v) > 0} \text{height}_v$
- initially $\phi = 0$
- $\phi$ increases by lifts in total at most $3n^2/2 - 5n/2$
- $\phi$ increases by saturating pushes at most $2n$ per push, in total $O(n^2 m)$
- $\phi$ decreases only by non-saturating pushes, and decreases at least one by a non-saturating push across $(v, u)$:
  - $v$ is desactivated
  - $u$ may activate after this push
  - $\text{height}_v \geq \text{height}_u + 1$
- $\rightarrow$ at most $O(n^2 m)$ non-saturating pushes
- $\rightarrow$ number of PUSH-REQUEST msg is $O(n^2 m)$

Message complexity is $O(n^2 m)$
**Time Complexity :** $O(n^2)$

**The first phase**
- node height is set by the earliest pulses
- number of pulses is the length of longest path from $t$ to $s$, so is $O(n)$

**The second phase**
- for any value of flow:
  - is transmitted straight toward $t$ → number of pulses = length(flow’s path) $< n$
  - is trapped at a node: make node lift → number of pulses = number of lifts of nodes on flow’s path
- number of pulses is $O(n^2)$
Summary

- A new asynchronous distributed algorithm
- Is refined to be an adaptative algorithm
- Can be improved the complexity

Open problem

- Minimum cost flows