ON THE IMPLEMENTATION OF GOLDBERG'S MAXIMUM FLOW ALGORITHM IN EXTENDED MIXED NETWORK

Nguyen Dinh Lau[1], Tran Quoc Chien[1], Phan Phu Cuong[2] and Le Hong Dung[3]

[1]University of Danang, Danang, Vietnam

[2]Vinaphone of Danang, Vietnam

[3]College of Transport II, Ministry of Transport, Vietnam

ABSTRACT

In this paper, we solve this problem of finding maximum flow in extended mixed network by Revised preflow-push methods of Goldberg This algorithm completely different algorithm postflow-pull in [15]. However, we share some common theory with [15].

KEYWORDS

Algorithm, maximum flow, extended mixed network, preflow, excess.

1. INTRODUCTION

In real life, we do not always have the freedom of choice that this idealized scenario suggests, because not all pair of reduction relationships between these problems have been proved, and because few optimal algorithms for solving any of the problem has yet been invente, and perhaps no efficient reduction that directly relates a given pair of problems has yet been devised.

In this paper, We consider another approach to solving the maximum flow problem in network mixed network. Using a generic method known as *preflow-push* method, we incrementally move flow along the outgoing edges of vertices that have more inflow than outflow. The *preflow-push* approach was developed by A. Goldberg and R.E. Tarjan in 1986 [4] on basis of various earlier algorithms. It is widely used because of its simplicity, flexibility.

As we did in augmenting-path algorithms, we use the residual network to keep track of the edges that we might push flow through. Every edge in the residual network represents a potential place to push flow. If a residual network edge is in the same direction as the corresponding edge in the flow network, we increase the flow; if it is in the opposite direction, we decrease the flow. If the increase fills the edge or the decrease empties the edge, the corresponding edge disappears from the residual network. For preflow-push algorithms, we use an additional mechanism to help decide which of the edges in the residual network can help us to eliminate active vertices.

The problem of finding maximum flow in network mixed network is extremely interesting and practically applicable in many fields in our daily life, especially in transportation. The paper develops a model of extended mixed network that can be applied to modelling many practical problems more exactly and effectively.

DOI:10.5121/ijcsit.2017.9609

Given a graph network G (V, E) with a set of vertices V and a set of edges E, where edges can be directed or undirected, with edge capacity ce:E→R*, so that ce(e) is adge capacity e ∈ E and vertices capacity cv:V→R*, so that cv(u) is vertices capacity u ∈ V.

With edge cost be be: E→R*, be(e): cost must be return to transfer an unit transport on edge e
With each v∈ V, Set Ev are set edge of vertice v.

Vertice cost bv:V×Ev×Ev→R*, bv(u,e,e'): cost must be return to transfer an unit transport from edge e to vertice u to edge e'.

A set (V, E, ce, cv, be, bv) is called extended mixed network.

2. FLOW EDGE ON EXTENDED MIXED NETWORK

The following formatting rules must be followed strictly. This (.doc) document may be used as a template for papers prepared using Microsoft Word. Papers not conforming to these requirements may not be published in the conference proceedings.

Given an extended mixed network G = (V, E, ce, cv, be, bv). where s is source vertex, t is sink vertex. A set of flows on the edges f = {f(x,y) | (x,y)∈E} is called flow edge on extended mixed network. So that:

(i) $0 \le f(x,y) \le ce(x,y)\ \forall (x,y) \in E$ (1)

(ii) For any vertex k is not a source or sink

$$\sum_{(v,k)\in E} f(v,k) = \sum_{(k,v)\in E} f(k,v) \quad (2)$$

(iii) For any vertex k is not a source or sink

$$\sum_{(v,k)\in E} f(v,k) \le cv(k) \quad (3)$$

The maximum problem:

Given an extended mixed network G(V, E, ce, cv, be, bv), where s is source vertex, t is sink vertex. The task required by the problem is finding the flow which has a maximum value. The flow value is limited by the total amount of the circulation possibility on the roads starting from source vertex. As a result of this, there could be a confirmation on the following theorem.

3. PREFLOW-PUSH METHODS

3.1. Some basic concept

◊ Residual extended network G_f:

For flow f on G = (V, E, *ce, cv, be, bv*), where s is source vertex, t is sink vertex. Residual extended network, denoted G_f is defined as the extended network with a set of vertices V and a set of edge E_f with the edge capacity is ce_f and vertices capacity is cv_f as follows:

- For any edge (u, v) ∈ E, if f(u, v)> 0, then (v, u) ∈ E_f with edge capacity is ce_f (v,u)=f(u, v)

- For any edge (u,v) ∈ E, if c(u,v) -f(u, v)> 0, then (u, v) ∈ E_f with edge capacity is ce_f(u,v) = ce(u,v) - f(u,v)

- For any vertices v∈ V then

$$cv_f(v) = cv(v) - \sum_{(x,v)\in E} f(x,v) \qquad (4)$$

◊ *preflow:*

For extended mixed network G = (V, E, *ce, cv, be, bv*). Preflow is a set of flows on the edges f = {f(x, y) | (x, y)∈ G} So that

(i) $0 \le f(x, y) \le ce(x, y)\ \forall (x, y) \in E$

(ii) for any vertex k is not a source or sink, inflow is not smaller than outflow, that is

$$\sum_{(v,k)\in E} f(v,k) \ge \sum_{(k,v)\in E} f(k,v) \qquad (5)$$

(iii) for any vertex k is not a source or sink

$$\sum_{(v,k)\in E} f(v,k) \le cv(k) \qquad (6)$$

Each vertex whose inflow is larger than its outflow is called the unbalanced vertex. The difference between a vertex's inflow and outflow is called excess. The concept of residual extended network G_f is similarly defined as flow.

The idea of this methods is balancing inflow and outflow at the balanced vertices by pushing along an outgoing edge and pushing against an incoming edge. Process of balancing is repeated until no more the unbalanced vertex then we get maximum flow. We store the unbalanced vertices on a generalized queue. A tool called a height function is used to help select the edge available in residual network to eliminate the unbalanced vertices. Now we assume that a set of the network is denoted as V={0,1,...,|V|-1}.

◊ *height function* of the pre-flow in the extended mixed network G = (V, E, *ce, cv, be, bv*), is a set of non-negative vertex weights h(0), ..., h(|V| −1) such that h(z) = 0 for z and h(u) ≤ h(v) + 1 for every edge (u,v) in the residual extended network for the flow. An eligible edge is an edge (u,v) in the residual extended network with h(u)=h(v) + 1.

A trivial height function is h(0) = h(1) = ... = h(|V| − 1) = 0. Then if we set h(a)= 1, any positive edge to a is the priority edge.

We define a more interesting height function by assigning to each vertex the latter's shortest – path distance to the sink (its distance to the root in any BFS tree of the network rooted at z. This height function is valid because h(z)= 0, and for any pair of vertices u and v connected by an edge (u,v) in residual network G_f, then h(u) ≤ h(v) + 1, because the path from u to z with edge (u,v) (h(v)+1 must be not shorter than the shortest path from u to z is h(u)).

3.2. Preflow-push algorithm

This is a particular algorithm in Preflow-push method. Here the unbalanced vertices are spushed into the queue. With each vertex from the queue, we will push the flow in the priority edge until

the flow becomes either balanced or does not have any priority edge. If it does not exist priority edge but there are unbalanced vertices, then we increase the height and push it into the queue. This algorithm inplements the genetic vertex- based preflow-push maxflow algorithm, using a generalized queue that disallows duplicates for active nodes.

The shortest-paths function is used to initialize vertex heights in the array h, to shortest- paths distances from the sink. The array contains each vertex's excess flow and therefore implicitly defines the set of active vertices. By convention, s is initially active but never goes back on the queue and t is nener active.

The main loop choose an active vertex v, then pushes flow through each of its eligible edges (adding vertices that receive the flow to the active list, if necessary), until either v become inactive or all its edges have been considered. In the latter case, v's height is incremented and it goes back into the queue.

Initially, the only preflow is in the edges for the source vertices is the following: $f(s,v)=min\{ce(s,v),cv(v)\}$ and other flows are 0. Then the first vertices of those edges directed to the source are unbalanced. With any unbalanced vertex v, we have (v, s) $\in E_f$ and (s,v) $\notin E_f$, inferred there exists paths from v to a, and there are no directed paths from source vertex a to sink vertex in the residual network G_f. So the properity is true with the initial flow.
Now we can describe the Preflow-push algorithm as follows:

Input: Extended mixed network G(V, E, *ce, cv, be, bv*). with source s, sink t.
Output: Maximum flow

$$F=(f_{ij}),(i,j)\in E$$

{
Initialize:
$\forall(u,x)\in E, f(u,x)=0$
$\forall(s,v)\in E$, $f(s,v)= min\{ce(s,v), cv(v)\}$
Select any available heigth function h in the extended mixed network G.
Q←u| u$\in V$ *and* $excess(u)\neq 0$.
While $Q!=\emptyset$ ***do***
{
Get u from Q.
If ∃ (u, v) ∈ E_f is priority edge then
{
Unbalanced vertex u:
If f(v,u)>0 then push along the edge (u,v) a flow with value $min\{excess(u),ce_f(u,v)\}$).
If $(u,v)\in E$ and $cv_f(v)>0$) then push along the edge (u,v) a flow with value
$min\{excess(u), ce_f(u,v), cv_f(v)\}$
If $excess(v)\ != 0$ then Q=Q∪ $\{v\}$.
}
Else
{
h(u)= 1 + min {h (v) | (u, v) ∈ E_f}
v →Q
}
}

```
        Prinf f is maximum flow
}
```

There are many optionsto explore in developing preflow-push implementations. We have already discussed three major choices:

- Edge-based versus vertex-based generic algorithm
- Generalized queue implementation
- Initial assignment of heights

If a vertex's height is greater then |V|, then there is no path from that vertex to the sink in the residual extended network G_f.

Theorem 3.1. The complexity of the preflow-push method is O ($|V|^2|E|$) [5].

Proof: We bound the number of phases using a potential function. This argument is a simple example of a powerful technique in the analysis of algorithm and data structures.

Define the quantity Q to be 0 if there are no active vertices and to be the maximun height of the active vertices otherwise, then consider the effect of each phase on the value of Q. Let $h_0(s)$ be the initial height of the source. At the beginning, $Q= h_0(s)$; at the end, Q=0.

First, we note that the number of phases where the height of some vertex increases is no more than $2|V|^2-h_0(s)$, since each of the V vertex heights can be increased to a value of at most 2|V|, by the corollary: *"Vertex's height is always less than 2.|V|"*.

While Preflow-push algorithm is in execution, there always exists a directed path from sink vertex to the unbalanced vertex in the residual extended network, and there are no directed paths from source vertex to sink vertex in the residual extended network.

We need to consider only unbalanced vertices, the height of each unbalanced vertex is either the same as or 1 greater than it was the last time that the vertex was balanced. By the same argument, the path from a source vertex to a given unbalanced vertex in the residual extended network G_f implies that unbalanced vertex's height is not greater than the source vertex's height plus |V| -2 (the sink vertex can not be on the path). Since the height of the source never changes, and it is initially not greater than |V|, the given unbalanced vertex's height is not greater than 2.|V| - 2, and no vertex has height 2|V| or greater.

Since Q can increase only if the height of some vertex increases, the number of phases where Q increases is no more than $2|V|^2- h_0(s)$.

If, however, no vertex's height is incremented during a phase, the Q must decrease by at least 1, sine the effect of the phase was to push all excess flow from each active vertex to vertices that have smaller height.

Together, these facts imply that the number of phases must be less than $4|V|^2$: The value of Q is $h_0(s)$ at the beginning and can be incremented at most $2|V|^2- h_0(s)$ times and therefore can be decremented at most $2|V|^2$ times. The worst case for each phase is that all vertices are on the queue and all of their edges are examined, leading to the stated bound on the total running time.

This bound is tight. The number of phases used by the preflow-push algorithm is proportional to $|V|^2$.

Example: Finding the max flow for the following extended mixed network G (Figure 1). The order of the vertices is 1, 2, 3, 4, 5, 6.

Table 1. Edge capacity

Edge	ce
(1,2)	10
(1,3)	9
(2,3)	5
(2,5)	7
(3,4)	7
(3,5)	6
(4,6)	10
(4,5)	5
(5,6)	9

Table 2. Vertex capacity

Vertex	1	2	3	4	5	6
Cv	∞	10	9	10	9	∞

Table 3. Height function

Edge	1	2	3	4	5	6
H	3	2	2	1	1	0

The steps of the algorithm described in the following figure:

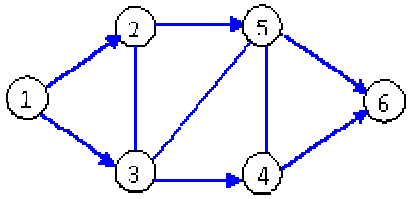

Figure 1. Network G

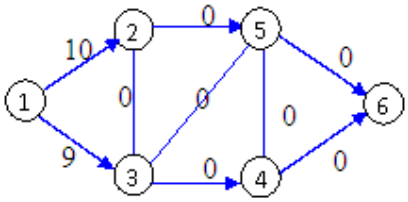

Figure 2. Initialze flow

Queue Q: > 3 | 2 >. Get unbalanced vertex 2 from the queue (Balanced 2):
Push along the edge (2, 5) a flow with value 7, We have preflow as in figure 3.

Table 4. Height function

Edge	1	2	3	4	5	6
h	3	4	2	1	1	0

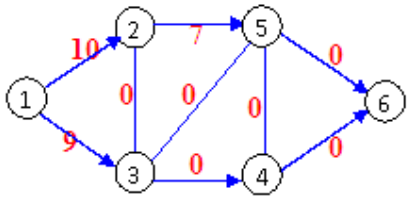

Figure 3. Preflow

We increase the heigth of the vertex 2 as follows: h(2) = 1 + 3 = 4. Height function change as in table 4

Push along the edge (2, 1) a flow with value 3, We have preflow as in figure 4.

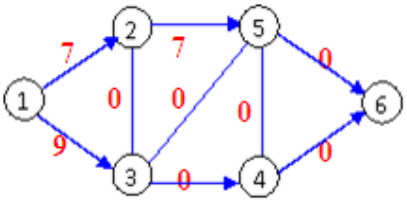

Figure 4. Preflow

Push unbalanced vertex 5 to Queue Q. Queue Q: >5 | 3 >
- Get unbalanced vertex 3 from the queue (Balanced 3):
Push along the priority edge (3, 4) a flow with value 7, We have preflow as in figure 5.

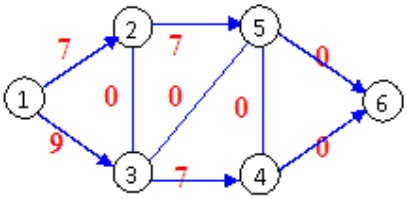

Figure 5. Preflow

Push along the priority edge (3, 5) a flow with value 2. We have preflow as in figure 6.

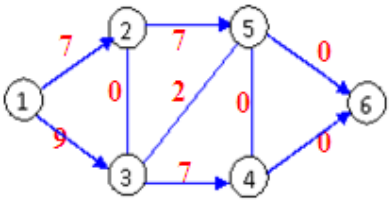

Figure 6. Preflow

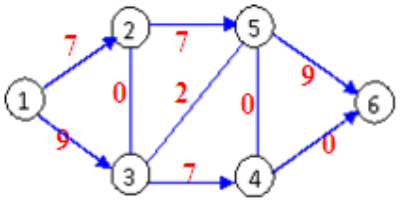

Figure 7. Preflow

Push unbalanced vertex 4 to Queue Q.
Queue Q: > 4 | 5>

- Get unbalanced vertex 5 from the queue (Balanced 5):
Push along the priority edge (5, 6) a flow with value 9, We have preflow as in figure 7.

- Get unbalanced vertex 4 from the queue (Balanced 4):
Push along the priority edge (4, 6) a flow with value 7, We have preflow as in figure 8.
Queue $Q = \varnothing$ (no more the unbalanced vertex), ended and the final flow is the maximum flow with value is 16

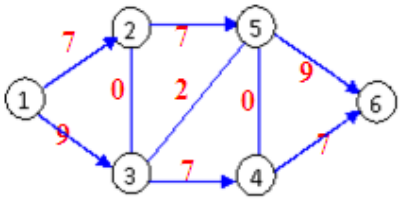

Figure 8. Preflow

4. BUILDING THE PARALLEL ALGORITHM

Inputs: Extended mixed network G with source s, sink t m processors (P_0, P_1,…, P_{m-1}), where P_0 is the main processor.

Output: Maximum flow

$$F = (f_{ij}), (i, j) \in E$$

Step 1: The main processor P_0 performs

(1.1). initialize: e, h, f, c_f, Q: set of unbalanced vertices (excluding the vertices s and t) are the vertices with positive excess.

(1.2). divide set of vertices V into sub-processors:

Let P_i be the i^{th} sub-processor (i = 1,2, ..., m-1)
P_i will receive the set of vertices V_i so that

$$(V_i \cap V_j = \phi \text{ if } i \neq j, \text{ and } \cup_i \{V_i\} = V)$$

(1.3). The main processor sends h, e, c_f to sub-processors

Step 2: The Condition to terminate: If Q = ∅, then postflow f becomes maximum flow, end. Else; choose unbalanced vertex u from Q. If exists priority edge (u, v) ∈ E_f then go to step 3, else go to step 4.

Step 3: m-1 sub-processors (P_1, P_2, …,P_{m-1}) implement

(3.1) Receive e, c_f, h and the set of vertices from the main processor

(3.2) Handling unbalanced vertice v (push and replace label). Get unbalanced vertexs u (e(u)>0) from Q and u∈ V_i (i= 1,2, ..., m-1).

If f(v,u)>0, then push along the edge (u,v) a flow with value $min\{delta,ce_f(u,v)\}$(where delta is the *excess* of the vertex u).

If $(u,v)\in E$ and $cv_f(v)>0$, then push along the edge (u,v) a flow with value $min\{delta, ce_f(u,v), cv_f(v)\}$

Step 4 : Increased the height of the vertex u as follows:

$$h(u) := 1 + \min \{h(v) \mid (u, v) \in E_f\}$$

Step 5: Send e, c_f, h to the main processor

Step 6: The main processor implements

(6.1) Receive e, c_f, h from step 5

(6.2) This step is distinctive from the sequential algorithms to synchronize our data, after receiving the data in (6.1), the main processor checks if all the edges $(u,v)\in E$ that have h(u)> h(v)+1, the main processor will relabel for vertices u, v as follows:

- e(u):= e(u)+ce_f(u,v), e(v):= e(v)-ce_f(u,v)
- If f(v,u)>0, then f(u,v):= $min\{delta,ce_f(u,v)\}$
  (where delta is the *excess* of the vertex u).
- If $(u,v)\in E$ and $cv_f(v)>0$,
  then f(u,v):= $min\{delta, ce_f(u,v), cv_f(v)\}$

Put the new unbalanced vertex into set Q

(6.3) If $\forall u \in V$ e(u)=0, eliminate u from active set Q.

Back to step 2.

Theorem 4.1. Postflow-pull parallel algorithm is true and has complexity O($|V|^2$ |E|).

Proof: Preflow-push parallel algorithm is build in accordance with other parallel computing system such as: PRAM, Cluster system, CUDA, RMI, threads,… Push and replace label using *atomic*, due to support of *atomic 'read-modify-write'* instructions, are executed atomically by the architecture. Other than the two execution characteristics provided by the architecture, we do not impose any order in which executions from multiple sub-processors can or should be interleaved, as it will be left for the sequential consistency property of the architecture to decide.

The outcome of the execution reduces to only a few simplified scenarios. By analyzing these scenarios, we can show that function f is maintained as a valid height function. A valid d guarantees that there does not exist any paths from s to t throughout the execution of the

algorithm, and hence guarantees the optimality of the final solution if the algorithm terminates. The termination of the algorithm is also guaranteed by the validatity of h, as it bounds the number of push and relabel operations to $O(|V|^2|E|)$.

5. CONCLUSIONS

The detail result of this paper is building sequential and parallel algorithm by preflow-push methods to find maximum flow in extended mixed network. The results of this paper are basically systematized and proven.

REFERENCES

[1] Chien Tran Quoc. Postflow-pull methods to find maximal flow. Journal of science and technology - University of DaNang, 5(40), 31-38, 2010.

[2] L. R. Ford and D. R. Fulkerson. Flows in Networks. Princeton University Press, 1962.

[3] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM, vol. 19, no. 2, pp. 248–264, 1972.

[4] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. Proceedings of the eighteenth annual ACM symposium on Theory of computing. New York, NY, USA: ACM, pp. 136-146, 1986.

[5] Robert Sedgewick. Algorithms in C Part 5: Graph Algorithms. Addison-Wesley, 2001.

[6] R. J. Anderson and a. C. S. Jo. On the parallel implementation of goldberg's maximum flow algorithm. Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures. New York, NY, USA: ACM, pp. 168–177, 1992.

[7] D. Bader and V. Sachdeva. A cache-aware parallel implementation of the push-relabel network flow algorithm and experimental evaluation of the gap relabeling heuristic. Proceedings of the 18th ISCA International Conference on Parallel and Distributed Computing Systems, 2005.

[8] B. Hong. A lock-free multi-threaded algorithm for the maximum flow problem. IEEE International Parallel and Distributed Processing Symposium, 2008.

[9] Zhengyu He, Bo Hong. Dynamically Tuned Push-Relabel Algorithm for the Maximum Flow Problem on CPU-GPU-Hybrid Platforms. School of Electrical and Computer Engineering-Georgia Institute of Technology, 2010.

[10] Chien Tran Quoc, Lau Nguyen Dinh, Trinh Nguyen Thi Tu. Sequential and Parallel Algorithm by Postflow-Pull Methods to Find Maximum Flow. Proceedings 2013 13th International Conference on Computational Science and Its Applications, pp 178-181, 2013.

[11] Lau Nguyen Dinh, Thanh Le Manh, Chien Tran Quoc. Sequential and Parallel Algorithm by Pre-Push Methods to Find Maximum Flow. Vietnam Academy of Science and Technology AND Posts & Telecommunications Institute of Technology, special issue works Electic, Tel, IT; 51(4A) pp 109-125, 2013.

[12] Lau Nguyen Dinh, Chien Tran Quoc and Manh Le Thanh. Parallel algorithm to divide optimal linear flow on extended traffic network. Research, Development and Application on Information & Communication Technology, Ministry of Information & Communication of Vietnam, No 3, V-1, 2014.

[13] Naveen Garg, Jochen Könemann. Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems. SIAM J. Comput, Canada, 37(2), pp. 630-652, 2007.

[14] Lau Nguyen Dinh, Chien Tran Quoc, Thanh Le Manh.. Improved Computing Performance for Algorithm Finding the Shortest Path in Extended Graph. proceedings of international conference on foundations of computer science (FCS'14), pp 14-20, 2014.

[15] Nguyen Dinh Lau, Tran Quoc Chien, Sequential and parallel Algorithm to find maximum flow on extended mixed networks by revised postflow-pull methods, Fourth International Conference on Advanced Information Technologies and Applications (ICAITA 2015), ISSN: 2231–5403, ISBN: 978-1-921987-43-4, DOI: 10.5121/csit.2015.51501, 2015, pp. 19-28