PRIORITY BASED FLOW IMPROVEMENT WITH INTERMEDIATE STORAGE

URMILA PYAKUREL[1] AND MOHAN CHANDRA ADHIKARI[2]

Central Department of Mathematics, Tribhuvan University,
PO Box 13143, Kathmandu, Nepal
Emails: [1]*urmilapyakurel@gmail.com and* [2]*mohan80700@gmail.com*

Abstract: Every models in the network flow theory aim to increase flow value from the sources to the sinks and reduce time or cost satisfying the capacity and flow conservation constraints. Recently, the network flow model without flow conservation constraints at the intermediate nodes has been investigated by Pyakurel and Dempe [13]. In this model, if the incoming flow to a intermediate node is greater than the outgoing flow, then the excess flow can be stored at the node respecting its capacity. Moreover, the model works if sum of the outgoing arc capacities from the source of a network is greater than the minimum cut capacity and intermediate node has storage capacity. The excess flow has been sent as far as possible from the source. In two terminal network, the maximum static flow and maximum dynamic flow problems with intermediate storage have been solved in polynomial time complexities.

Motivated by this work, we study the lexicographic maximum flow problem with intermediate storage in single source and multiple sink networks by assigning priority ordering. The problem is to maximize the flow value at each sink in fixed priority ordering and push the excess flow from the source as far as possible to the intermediate nodes. We present polynomial time algorithms to solve it in both static and dynamic networks.

Key Words: Network flow, priority ordering, lex-maximum flow, intermediate storage.

AMS (MOS) Subject Classification. Primary: 90B10, 90C27, 68Q25; Secondary: 90B06, 90B20.

1. Introduction

Network flow theory is applicable to solve various real life problems. One interesting example is the evacuation planning. After disaster (man made or natural) saving lives from accidental areas (sources) to safe places (sinks) as quickly and efficiently as possible is very important. The streets between sources and sinks are considered as arcs that have integer capacities and cost or transit times. The intersections of the streets are the intermediate vertices that have integer capacities. The group of evacuees passing through the network is a flow which is to be maximized using existing network capacities. According to different evacuation models, movement towards the source is not allowed, so road (arcs) towards the sources remain unused and can be used to increase the flow towards the sinks. The empty arc capacities are managed using different models and algorithms as presented in [12, 17, 18, 20, 19]. An overview of different approaches to solve the evacuation planning problem can be found in Dhamala et al.[2]. The network flow theory with intermediate storage is also the problem of empty arc management so that a maximum flow can be sent out from the source and pushed them as far as possible by allowing excess flow storage at the intermediate vertices, [13].

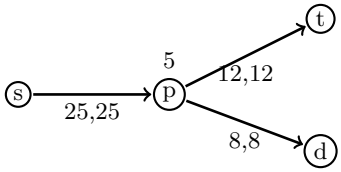
Corresponding author: Urmila Pyakurel .

The maximum flow model introduced by Ford and Fulkerson [4, 5] maximizes the flow gradually from the source to the sink satisfying the feasibility and flow conservation constraints. According to this model, maximum flow in the network is equal to the minimum cut capacity. Preflow push algorithm developed by Goldberg and Tarjan [6] is a counterpart of the augmenting path algorithm of [4, 5] that violates conservation constraints at the intermediate vertices and pushes excess flow using single arc toward the vertices that are closer to the sinks. Maximum flow obtained in a network with priority ordering to sinks or sources is lexicographic maximum flow. We send as much flow as possible to the sink with the first priority. This value is fixed and again we send maximum flow to the sink with the second priority and so on. Minieka [11] introduced the lexicographic maximum static flow problem in a multiple source multiple sink network and solved it in polynomial time complexity. Later, Hoppe and Tardos [7, 8] introduced the lexicographic maximum flow problem in dynamic network with given priority ordering of the sources and the sinks and presented a polynomial time algorithm to solve it in the original network. Similarly, Pyakurel and Dhamala [17] solved the lexicographic maximum static and dynamic flow problems allowing arc reversal capability in polynomial time complexity. Kamiyama [9] investigated the lexicographic maximum flow at every time point and presented a pseudo-polynomial time algorithm to solve it. All these problems are solved with flow conservation at each intermediate vertex. They do not care the excess arc capacities as well as the capacity of intermediate vertices.

If the inflow into an intermediate vertex is more than outflow, then there will be an intermediate storage. The possibility of intermediate storage, by placing emergency units on intermediate vertices, to reduce evacuation time had been discussed by Pyakurel and Dhamala [16, 15]. Recently, Pyakurel and Dempe [13] investigated the network flow models with intermediate storage that is suitable if sum of the outgoing arc capacities from the source is greater than the minimum cut capacity of the network. The flow value equal to minimum cut capacity reaches to the sinks and excess is stored at intermediate vertices. The model assumes intermediate vertices have enough capacities with lower bound bounded by the sum of arc capacities incoming to the vertices. It always gives the first priority to the sinks with the assumptions that the vertices at the longest distance from the source are more safer. The major contribution of their model is to increase the flow value leaving the source using full capacity of arcs.

They introduced the maximum static and dynamic flow problems with intermediate storage on two terminal networks and presented polynomial time algorithms to solve them. Moreover, they investigated the arc reversals approach for this model and introduced the maximum dynamic flow problem with arc reversals capability. They solved the problem by presenting an algorithm with polynomial time complexity. Moreover, they showed that the earliest arrival flow with intermediate storage on two terminal series parallel network can be solved in strongly polynomial time complexity, (see also [14] for detail). They studied the effect of lane reversals to the problem and solved it with the same complexity. Khadka and Bhandari [1, 10] also considered the non-conservative aspect of the network flow problem, however, a theoretical validity proof and an experimental verification are lacking.

In order to show the importance of the intermediate storage, we consider an example with source s, intermediate vertex p and two sinks d and t as in figure. Two paths s-p-t and s-p-d carry 12 and 8 flow units, respectively if we do not allow intermediate storage. By using excess capacity from s to p, we can send 5 units more when we allow intermediate storage. If s is more risk area and we have to make empty it as soon as possible, then excess flow storage at intermediate vertices with sufficient capacity is very important.



Authors in [13, 14] used the lexicographic properties of [11] and [8] to solve the static and dynamic maximum flow problems with intermediate storage in single source and single sink network, respectively. In this paper, the lexicographically maximum flow problem with intermediate storage is investigated independently. We present a polynomial time algorithm that gives the three layers of priority ordering. We introduce

the lex-maximum static and dynamic flow problems with intermediate storage having fixed priority ordering and solve them with polynomial time algorithms in the original network.

The organization of the paper is as follows. In Section 2, we develop some mathematical notations, give network formulations and describe the flow models with intermediate storage. We introduce the lex-maximum static flow problem with intermediate storage in Section 3 and present a polynomial time algorithm. The lex-maximum dynamic flow problem with intermediate storage is introduced and solved it with a polynomial time algorithm in Section 4. This paper is concluded in Section 5.

2. Preliminaries

Let V be the set of n vertices, A be the set of m arcs, s be a single source and $I = \{h_y : y = 1, 2, \ldots, l\}$ and $D = \{d_x : x = 1, 2, \ldots, r\}$ be the set of intermediate vertices and multiple sinks, respectively. Each arc $e = (i, j) \in A$ is associated with the capacity function $u : A \to \mathcal{Z}^+$ which determines the maximum amount of flow that is to be transship from vertex i to j. For each intermediate vertex $h \in I$, the function $v : I \to \mathcal{Z}^+$ determines the holding capacity that permits maximum amount of flow to be stored. With these data, we obtain a static network $\mathcal{N} = (V, A, u, v, c, s, I, D)$ in which each arc has a function $c : A \to \mathcal{Z}^+$ that gives required cost for one unit of flow from vertex i to j along the arc (i, j). If the cost is considered as time function on each arc as $\tau_e : A \to \mathcal{Z}^+$ gives the time required to travel flow along the arc $e = (i, j)$, then the network is dynamic one. The transit time means if one unit of flow is pushed from the vertex i at time θ along the arc (i, j) after $\theta+\tau_e$ time it reaches vertex j. Sometime we are given a predetermined time T with in which all the flow should be shifted from a source to the sinks. Our dynamic network is $\mathcal{N} = (V, A, u, v, s, I, D, \tau, T)$ with the set of discrete time $\mathbf{T} = \{0, 1, 2, \ldots, T\}$. For any vertex i, the set of incoming and outgoing arcs are denoted as: $A^i_\uparrow = \{e \in A : e = (j, i),\ j \in V\}$ and $A^i_\downarrow = \{e \in A : e = (i, j),\ j \in V\}$ respectively. In general, no arcs enter the source and exit from the sinks so $A^s_\uparrow = \emptyset$ and $A^D_\downarrow = \emptyset$

2.1. Flow models.

All the flow models without intermediate storage do not address the problem of using excess capacity than the minimum cut. If total amount of flow outgoing from the source is more than the capacity of minimum cut then all flow can not reach the sinks. The excess flow either do not leave the source or hold at intermediate vertices so that we can push back to the source. Here, we consider the flow models with intermediate storage presented in [13] that use the excess capacity of arcs and vertices to push flow as maximum as possible from the source.

Let $f : A \to \mathcal{R}^+$ be a static flow of value $val(f)$ and $f(h) : I \to \mathcal{Z}^+$ be the amount of flow stored in the intermediate vertices. The objective is to maximize

$$val(f) = \sum_{e \in A^s_\downarrow} f(e) = \sum_{e \in A^D_\uparrow} f(e) + \sum_{h \in I,\ v(h) > 0} f(h) \tag{2.1}$$

in which total amount of flow outgoing from the source is equal to the sum of total amount of flow reaching to the sinks and amount of flow stored at intermediate vertices. Notice that the amount of flow that reaches the sinks is equal to the minimum cut capacity and therefore it is the flow value given by the flow models without intermediate storage, [5].

Now, Objective 2.1 is to be maximized under the flow conservation constraints with intermediate storage 2.2 together with the feasibility condition on the arcs and vertices given by 2.3 and 2.4.

$$f(h) = \sum_{e \in A^i_\uparrow} f(e) - \sum_{e \in A^i_\downarrow} f(e) \quad \geq 0 \quad , \ \forall h \in I \tag{2.2}$$

$$0 \leq f(e) \quad \leq \quad u(e), \quad \forall e \in A \tag{2.3}$$

$$0 \leq f(h) \quad \leq \quad v(h), \quad \forall h \in I \tag{2.4}$$

A dynamic flow g of value $val(g, T)$ in discrete time $\mathbf{T}$ is a function $g : A \times \mathbf{T} \to \mathcal{R}^+$. A maximum dynamic flow with intermediate storage maximizes the value $val(g, T)$ given in the objective function 2.5

satisfying the conservation constraints with intermediate storage 2.6 and the feasibility constraint on arcs 2.7 and vertices 2.8,[13, 14].

$$val(g,T) = \sum_{e\in A^s_{\downarrow}} \sum_{\sigma=0}^{T} g(e,\sigma) = \sum_{e\in A^D_{\uparrow}} \sum_{\sigma=\tau_e}^{T} g(e,\sigma-\tau_e) \quad + \quad \sum_{h\in I,\ v(h)>0} g(h,T) \tag{2.5}$$

$$g(e,\theta) = \sum_{e\in A^i_{\uparrow}} \sum_{\sigma=\tau_e}^{\theta} g(e,\sigma-\tau_e) - \sum_{e\in A^i_{\downarrow}} \sum_{\sigma=0}^{\theta} g(e,\sigma) \quad \geq \quad 0, \quad \forall h \in I,\ \theta \in \mathbf{T} \tag{2.6}$$

$$0 \leq g(e,\theta) \quad \leq \quad u(e,\theta), \quad \forall e \in A \ \ \theta \in \mathbf{T} \tag{2.7}$$

$$0 \leq g(h,\theta) \quad \leq \quad v(h,\theta), \quad \forall h \in I \ \ \theta \in \mathbf{T} \tag{2.8}$$

where $g(h,\theta) : I \times \mathbf{T} \to \mathcal{Z}^+$ is the amount of flow storage in intermediate vertices in time θ.

When we wish to maximize the flow at each time period $\theta \in \mathbf{T}$, the flow is earliest arrival flow (EAF) with value:

$$val(g,\theta) = \sum_{e\in A^s_{\downarrow}} \sum_{\sigma=0}^{T} g(e,\sigma) \quad \geq \quad \sum_{e\in A^D_{\uparrow}} \sum_{\sigma=\tau_e}^{T} g(e,\sigma-\tau_e) \tag{2.9}$$

satisfying the conditions 2.6, 2.7 and 2.8

3. Lex-maximum static flow with intermediate storage

In this section, we investigate the lex-maximum static flow problem with intermediate storage. Before this, we discuss the procedure to find priority ordering in our network.

We have given s-D network $\mathcal{N} = (V, A, u, v, c, s, I, D)$. First we fix the priority ordering of all $i \in V\backslash\{s\}$ in three layers. We assume that flow can not be stored at intermediate vertices if their holding capacity is less than the sum of the capacity of incoming arcs. For this, we explore the ordering according to Algorithm 3.1. First layer gives the priority order for sinks $d_x \in D, x = 1, 2 \ldots, r$ and second layer for all $h_y \in I, y = 1, 2, \ldots, l$ with $v(h) \geq \sum_{e\in A^i_{\uparrow}} u_e > 0$ separately. Third layer combines previous two layers giving first priority to first layer and second priority to second layer. We adopt the concepts of [13] to find the priority ordering.

Algorithm 3.1. *Three layers for priority ordering*

Input: *Network* $\mathcal{N} = (V, A, u, v, c, s, I, D)$

(1) *First layer priority ordering*
- *For each sink* $d_x \in D$*, determine the minimum distance* $dis(s, d_x)$*,* $x = 1, 2, \ldots, r$*.*
- *If* $dis(s, d_1) < dis(s, d_2) < \cdots < dis(s, d_r)$*, set priority ordering as* $(d_r, d_{r-1}, \ldots, d_2, d_1)$*.*
- *If* $dis(s, d_j) = dis(s, d_k)$ *for some* $j \neq k$*, set priority order arbitrarily.*

(2) *Second layer priority ordering*
- *For each intermediate vertex* h *with* $v(h) > 0$*, determine the minimum distance* $dis(s, h_y)$ *for* $y = 1, 2, \ldots, l$*.*
- *If* $dis(s, h_1) < dis(s, h_2) < \cdots < dis(s, h_l)$*, set priority ordering as* $(h_l, h_{l-1}, \ldots, h_2, h_1)$*.*
- *If* $dis(s, h_j) = dis(s, h_k)$ *for some* $j \neq k$*, set priority order arbitrarily.*

(3) *Third layer priority ordering*
- *Set first priority ordering to layer* (1) *and then layer* (2)*.*

Output: *Priority ordering*

Lemma 3.2. *Algorithm 3.1 computes priority ordering in polynomial time.*

Proof. In the first layer the minimum distance of sinks from source can be obtained in $O(n^2)$ times using Dijkstra algorithm [3] and their priority ordering can be obtained in linear time. Similarly second layer can be performed with same complexity as the first layer. Finally, the third layer operation requires only the linear time. Thus, computing priority ordering can be performed in polynomial time complexity. □

Definition 3.3. If we have $D_1 \subseteq D_2 \subseteq D_3 \subseteq \cdots \subseteq D_r \subseteq D$, then the lexicographically maximum flow on the sinks with intermediate storage is the maximal flow that delivers $V(D_x)$ units into each of the subsets D_x, for $x = 1, 2, \ldots, r$ and $V(I_y)$ units flow that does not reach the sinks will be stored in the intermediate vertices I_y for $y = 1, 2, 3, \ldots, l$ where $r + l < n$.

In order to solve the problem, we have to fix the priority of multiple sink of our network using first layer priority ordering of Algorithm 3.1. With this priority ordering, we can solve the lex-maximum static flow problem without intermediate storage using algorithm of [11]. As we are solving the problem with intermediate storage, we fix the priority ordering of intermediate vertices using the second layer priority ordering of Algorithm 3.1. Then, we maximize the flow in two level, i.e., with the first layer priority ordering in first level, and with the second layer priority ordering in second level. But the intermediate vertices are not sinks. They have only sufficient capacities to store flow. Then, we consider these vertices as virtual shelter with same capacities and priority ordering.

For each vertex $h \in I$, with $v(h) > 0$, we create a virtual shelter (vertex) h' and a virtual arc (h, h') with capacity $u(h, h') = v(h) = v(h')$ and $c(h, h') = 0$ time (cost). Virtual vertices $h' \in I'$ are considered as sinks and follow the priority ordering of the corresponding vertices $h \in I$. With the new virtual vertices and arcs the network is transformed as $\mathcal{N}' = (V', A', u', v', c, s, I, D')$ where $V' = V \cup I'$, $A' = A \cup \{(h, h')\}$, $u'(i, j) = u(i, j) \cup \{u(h, h')\}$, $v'(h) = v(h)$, and $D' = D \cup I'$. Now we have the priority ordering

$$\{d_1\} \subset \{d_1, d_2, \} \subset \cdots \subset D \subset D \cup \{h'_1\} \subset D \cup \{h'_1, h'_2\} \subset \cdots \subset D \cup I' = D'.$$

Thus, our network is converted into a single source multiple sink network with priority ordering. Now we can adopt the algorithm of [11] to solve the lex-maximum flow problem without intermediate storage. Let $\text{val}_{max}(D')$ be the total amount of flow that can enter the sinks D'. Then, $\text{val}_{max}(D')$ is the sum of the flows that goes from source to the each sinks $D \cup I$, which is the lexicographically maximum static flow (LMSF) on the sinks that satisfies the flow conservation constraints at the intermediate vertices as well as feasibility condition. With this solution, the virtual sinks and arcs are removed from the network and flows that are on virtual sinks are shifted to the corresponding intermediate vertices. Hence we obtain the LMSF with the intermediate storage. For more detail, we present Algorithm 3.4.

For the simplicity, we may assume that each intermediate vertices $h \in I$ with $v(h) > 0$ are acting as virtual sinks $h' \in I'$ with the respective priority ordering where I' is the set of virtual sinks.

Algorithm 3.4. *Lex-maximum static flow with intermediate storage*

Input: *Network* $\mathcal{N} = (V, A, u, v, c, s, I, D)$

(1) *Fix priority ordering using Algorithm 3.1.*

(2) *Construct transform network* $\mathcal{N}' = (V', A', u', v', s, I, D')$ *with virtual sinks.*

(3) *Compute the maximum flow at each sinks of* D' *in the priority ordering of Step 1.*

(4) *Transform the flow in original network by removing the virtual vertices and arcs.*

Output: *Lex-maximum static flow with intermediate storage on* $\mathcal{N}$.

Lemma 3.5. *Algorithm 3.4 always gives the feasible solution to the lex-maximum static flow problem with intermediate storage.*

Proof. Algorithm 3.1 always gives the feasible solution, so, Step 1 of Algorithm 3.4 is feasible. As the virtual sinks have same priority ordering and capacities of their corresponding intermediate vertices, and virtual arc capacities are bounded by vertex capacities, the construction of transform network is feasible in Step 2. Step 3 always gives feasible maximum flow at each sink according to [11], so it is feasible. At last, the transformation of solution from virtual network to the original network does not violate the feasibility constraints. Thus, our Algorithm 3.4 is feasible. □

Theorem 3.6. *An optimal solution to the lex-maximum static flow problem with intermediate storage can be obtained in polynomial time.*

Proof. We prove this theorem in two steps using Algorithm 3.4. First we show its feasibility verification and then its optimality proof. From Lemma 3.5, Algorithm 3.4 is feasible.

Now we have to prove the optimality of the algorithm. Step 1 fixes the priority ordering of sinks as well as intermediate vertices in polynomial time complexity. For the existence of flow conservation constraints at the intermediate vertices, we consider the intermediate vertices $h \in I$ as virtual sinks $h' \in I'$ with respective priorities and construct the transform network $\mathcal{N}' = (V', A', u', v', c, s, I', D')$ in linear time as in Step 2. Then, we compute the priority based maximum flow at each sinks in D' using algorithm of [11]. At first, we consider a sink with first priority and use a maximum flow algorithm to get maximum flow satisfying flow conservation constraints. Considering this flow as initial flow, we obtain the residual network where capacity is defined as:

$$u_f(i,j) = \begin{cases} u(i,j) - f(i,j) & \text{for all} \quad (i,j) \in A \\ f(i,j) & \text{for all} \quad (j,i) \in A \end{cases}$$

Flow is augmented along the $s - D'$ paths in residual network and gradually increases in each iteration. If there are no flow augmenting paths for the current sink, the obtained flow is maximum [4]. Next time, we consider the sink of the second priority and follow the same procedure. We solve the maximum flow problem at each sink with its priority ordering. Hence we obtained lex-maximum static flow on the transformed network without intermediate storage. Finally we transfer the flow accumulated in the virtual vertices to the corresponding intermediate vertices as in [13] that gives the lexicographic maximum flow with intermediate storage. The transformation can be done in liner time.

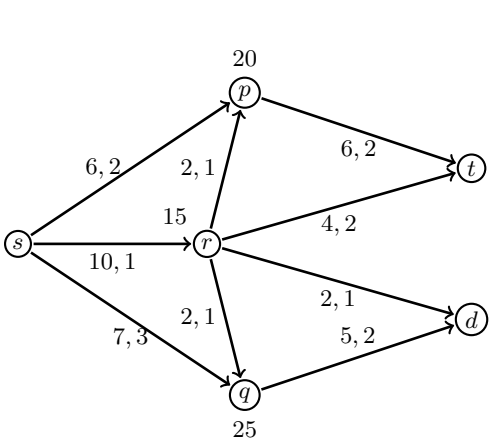
The complexity of the algorithm is determined by Step (3) as Steps (1), (2) and (4) can be obtained in polynomial time. With the solution procedure of [11], Step (3) can be obtained in polynomial time. Thus, the time complexity of Algorithm 3.4 is polynomial time. □

Example 3.7. We take a single source multiple sink network as in Figure 1 (a) where s is the source, $D = \{t, d\}$ is set of sinks and the set of intermediate vertices $I = \{p, q, r\}$. The set of arcs is $A = \{(s,p), (s,q), (s,r), (r,p), (r,q), (r,d), (r,t), (p,t), (q,d)\}$. Assume that p, q and r have capacity 20, 25 and 15 units whereas source and sinks t, d have sufficient capacities. Each arc is associated with capacity and cost, for example, arc (s,p) has capacity 6 and cost 2 per unit flow. Outgoing arcs from source can send 23 units flow which is more than the minimum cut capacity 17, so lexicographic maximum flow with intermediate storage is acceptable.

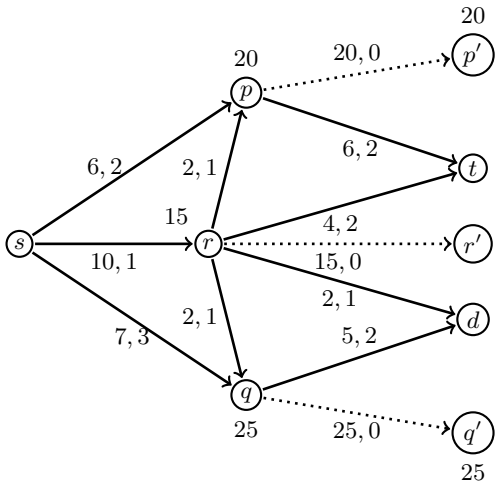
The minimum distance of sinks from the source are $dis(s,d) = 2$ and $dis(s,t) = 3$. Since $dis(s,t) > dis(s,d)$, sink t is in first priority and then d. On the same way, intermediate vertices have priority order q, p and r. To apply maximum flow algorithms, we set virtual shelters (considered as sinks) p', q' and r' for the intermediate vertices p, q and r with the same holdover capacity and priority ordering. The virtual network has the set of vertices $V' = V \cup I'$ where $I' = \{p', q', r'\}$, $A' = A \cup \{(p,p'), (q,q'), (r,r')\}$ and $D' = D \cup I'$ and s is the source.

We consider a super sink d^* and connect each sink with it by an infinite capacity arc at zero time(cost) as in Figure1 (c). Lexicographic maximum flow based on priority ordering is calculated with the solution procedure of [11].

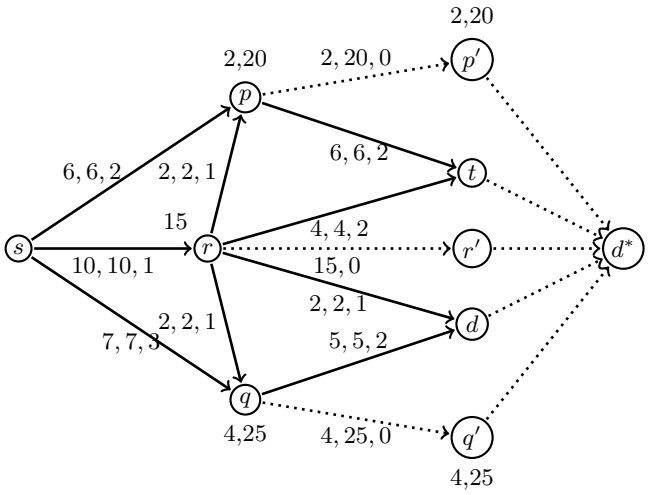
The sinks t and d receive 10 and 7 units flow. Similarly, the virtual sinks q' and p' receive 4 and 2 units flow that is shifted to q and p. Hence, maximum flow out of the source is 23 units out of which 17 units reaches the sinks and 6 units is stored at intermediate vertices.



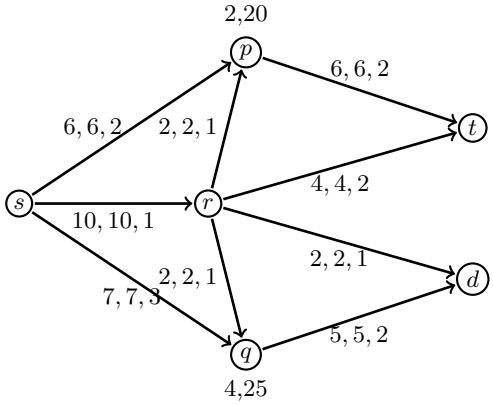
(a) (Arc: capacity, cost; vertex: capacity)



(b)



(c) (Arc: flow, capacity, cost; Vertex: flow, capacity)



(d)

FIGURE 1. (a) Given network (b) Network with virtual vertices (c) Transformed network (d) Lex-maximum flow with intermediate storage

4. Lex-maximum dynamic flow with intermediate storage

Flows associated with the time parameter in network are dynamic where each arc is associated with travel time together with capacity. Flow requires time to pass over the arcs and arc capacities changes over time. For a given time period T, the maximum dynamic flow problem is to obtain maximum value $val(g, T)$. But the lex maximum dynamic flow (LMDF) problem is to obtain the feasible flow that lexicographically maximizes the amount of flow in each of the terminals in priority basis. The LMDF problem was introduced and solved polynomially in which flow is conserved at each vertex [8].

In this section, we address LMDF problem with intermediate storage in a single source multiple sink network $\mathcal{N} = (V, A, u, v, s, I, D, \tau, T)$.

Definition 4.1. Let $\mathcal{N} = (V, A, u, v, s, I, D, \tau, T)$ be a dynamic network. The LMDF problem with intermediate storage on $\mathcal{N}$ is to find the feasible maximum flow at each sinks with fixed priority ordering and pushes the excess flow as far as possible from the source on the intermediate verticess respecting their capacities.

Recall that Pyakurel and Dempe [14, 13] used the lex-maximum dynamic flow algorithm to solve the maximum dynamic flow and earliest arrival flow problems with intermediate storage on two terminal

general network and series parallel network, respectively. Here, we are presenting independent lex-maximum dynamic flow algorithm, Algorithm 4.2 to solve the LMDF problem with intermediate storage. However, we adopt their solution technique to solve it.

First, we fix priority ordering of sinks and intermediate vertices as in Algorithm 3.1. The network transformation with arc and vertex capacities are carried out as in static case. Transit time on the virtual arc $\tau(h, h')$ is assumed to be zero and the vertex capacity is $v(h) \geq \sum_{e \in A^i_\uparrow} u_e > 0$. On the transformed network, we solve the lex-maximum dynamic flow problem without intermediate storage by using algorithm of [8]. Then virtual arcs and vertices are removed to transfer the flow in to original network that gives LMDF with intermediate storage.

Algorithm 4.2. *Lex-maximum dynamic flow with intermediate storage*

Input: *Network* $\mathcal{N} = (V, A, u, v, s, I, D, \tau, T)$

(1) *Set priority ordering using Algorithm 3.1.*

(2) *Construct transform network* $\mathcal{N}' = (V', A', u', v', s, D', \tau, T)$ *with virtual sinks.*

(3) *Compute the maximum flow at each sinks of* D' *in the priority ordering of Step 1 using algorithm of* [8].

(4) *Transform the flow in original network by removing the virtual vertices and arcs.*

Output: *Lex-maximum dynamic flow with intermediate storage on* $\mathcal{N}$.

Before we prove the correctness of our algorithm, we overview shortly how the algorithm of [8] works at Step 3. The lex-maximum dynamic flow algorithm of [8] gives LMDF without intermediate storage in polynomial time complexity. Its implementation procedure is as follows: introduce a super terminal vertex s^* and join it to each of the terminals $s_i,\ i = 1, 2, \ldots, k$ such that $u(s^*, s_i) = \infty$ and $\tau(s^*, s_i) = 0$. Consider this network as $\mathcal{N}^{k+1}$ where g^{k+1} is a zero flow. Now for each iterations $i = k, k-1, \ldots, 2, 1$, we choose a terminal s_i. If s_i is a source remove the arc (s^*, s_i) to get a network $\mathcal{N}^i$ and determine min cost max flow from s^* to s_i. If s_i is a sink add an arc s_i, s^* with $u(s_i, s^*) = \infty$ and transit time $-(T + 1)$. Mark this network as $\mathcal{N}^i$ and find min cost circulation f^i. In both cases consider time as cost. Update g^i as $g^{i+1} + f^i$. If β_i is the standard chain decomposition of f_i, then the set of chains $\alpha_i = \alpha_{i+1} + \beta_i$ gives the accumulated chain flows. The LMDF is achieved when $\alpha = \alpha_1$.

Theorem 4.3. *An optimal solution to the LMDF problem with intermediate storage can be obtained in polynomial time.*

Proof. Algorithm 4.2 is feasible since all the steps are feasible. Now, we prove its optimality. The complexity for Priority ordering of sinks and intermediate vertices in Step 1 is polynomial. Step 2 constructs the transform network $\mathcal{N}' = (V', A', u', v', s, I, D', \tau, T)$ in linear time. The transformed network contains virtual sinks $h' \in I'$ for each $h \in I$ so the intermediate vertices satisfy flow conservation constraints. Now, our network is reduced to the multi-sink network with priority ordering. In this transformed network, we use the algorithm of [8] and compute priority based maximum flow at each of the sinks D'. Min cost circulation is applied to the residual network repeatedly k times, where k is the number of terminals. Hence, lex-maximum dynamic flow is obtained in transformed network without intermediate storage in $O(k \times (m \log n(m + n \log n)))$ time complexity, where $O((m \log n(m + n \log n)))$ is the complexity to run a minimum cost circulation. Finally, the flow accumulated at virtual sinks is shifted to corresponding intermediate vertices as in [13]. At original sinks, the obtained flow is equal to the LMDF and excess flow is storage at intermediate vertices. Thus we obtain the LMDF with intermediate storage.

Steps 1, 2 and 4 can be accomplished in polynomial time, so Step 3 determines the complexity of our algorithm. Algorithm in [8] computes Step 3 in polynomial time. Hence, Algorithm 4.2 has polynomial time complexity. □

Example 4.4. Let us consider a network in Figure 1 (a) with $T = 5$. LMDF can be obtained with Algorithm 4.2 as follows. For each intermediate vertices virtual shelters (sinks) are created with same priority ordering and convert the network into a single source single sink as in static case. Now, we obtain the LMDF without

intermediate storage using Step (3) of Algorithm 4.2. Sinks t and d receive 24 and 15 units flow. Similarly, virtual sinks q', p' and r' receive 22, 20 and 14 units flow that are shifted the the original vertices q, p and r to obtain LMDF with intermediate storage.

5. Conclusions

We have studied the network flow theories from literature satisfying flow conservations as well as with intermediate storage. We considered a single source multiple sink network and investigated the flow models with intermediate storage for priority based maximum flow problem. We focused our study on two issues: setting priority ordering to the sinks and intermediate vertices, and obtaining maximum flow in the network with intermediate storage with that priority ordering. We developed an algorithm to determine the priority ordering of the sinks and intermediate vertices. If the outgoing arc capacities of a network is greater than the minimum cut capacity, the excess arc capacities can be used to push extra flow from the source to wards the intermediate vertices. To address this problem, we investigated lex-maximum static problem with intermediate storage and presented polynomial time algorithm to solve it in single source multiple sink network. Similarly, we introduced the lex-maximum dynamic flow problem with intermediate storage in the network and presented polynomial time algorithm to solve it.

To the best of our knowledge, we have introduced and solved these problems for the first time. Further, we are interested in the flow management stored at the intermediate vertices. This approach plays a vital role in evacuation as additional evacuees can be shifted from the disastrous areas.

Acknowledgments

The first author thanks Alexander von Humboldt Foundation for her return fellowship (November 2019-October 2020) at Central Department of Mathematics, TU, Nepal.

References

[1] Bhandari, P.P. and Khadka, S.R. (2019). Maximum flow evacuation planning problem with non-conservative flow constraints at the intermediate nodes. *International Conference on Mathematical Optimization (April 8-13,2019, Beijing).*

[2] Dhamala, T.N., Pyakurel, U. and Dempe, S. (2018). A critical survey on the network optimization algorithms for evacuation planning problems. *International Journal of Operations Research*, **15**(3), 101-133.

[3] Dijkstra, E.W. (1959). A note on two problems in connection with graphs. Numerische mathematik, **1**(3), 269-271.

[4] Ford, F.R. and Fulkerson, D.R. (1956). Maximal flows through a network. *Canadian Journal of Mathematics,* **8,** 399 – 404.

[5] Ford, F.R. and Fulkerson, D.R. (1962). *Flows in Networks.* Princeton University Press, Princeton, New Jersey, USA.

[6] Goldberg, A.V. and Tarjan, R.E. (1988). A new approach to maximum flow problem. *Journal of the Association for Computing Machinery* **35** (4): 753-782.

[7] Hoppe, B. and Tardos, E. (1994). Polynomial time algorithm for some evacuation problems. *Proc. of 5th Ann ACM-SIAM Symp on discrete algorithms,* Pages 433-441.

[8] Hoppe, B. and Tardos, E. (2000). The quickest transshipment problem. *Mathematics of Operations Research,* **25**:36-62.

[9] Kamiyama, N. (2019). Lexicographically optimal earliest arrival flows. *Networks*, **75**:18-33

[10] Khadka, S.R. and Bhandari, P.P. (2019). Model and Solution for Non-conservation Flow Evacuation Planning Problem. *The Nepal Math. Sc. Report*, 36(1& 2), 11-16.

[11] Minieka, E. (1973). Maximal, lexicographic, and dynamic network flows. *Operations Research*, **21**, 517-527.

[12] Nath, H.N., Pyakurel, U., Dhamala, T.N., and Dempe, S. (2020). Dynamic network flow location models and algorithms for evacuation planning. *Journal of Industrial and Management Optimization*, doi:10.3934/jimo.2020102.

[13] Pyakurel, U. and Dempe, S. (2019). *Network flow with intermediate storage: Models and algorithms.* Preprint, 01/2009 TU Bergakakademie Freiberg.

[14] Pyakurel, U. and Dempe, S. (2019). Universal maximum flow with intermediate storage for evacuation planning. *4th International conference on Dynamics of Disasters* July 1-4, Kalamata, Greece.

[15] Pyakurel, U. and Dhamala, T.N. (2012). *Contraflow emergency evacuation by earliest arrival flow.* PhD progress report presentation at University Grant Commission Nepal. $www.researchgate.net/profile/Urmila_Pyakurel3/publications$.

[16] Pyakurel, U. and Dhamala, T.N. (2012). Emergency facility location on contraflow transportation network for evacuation planning. *International Conference on Operations Research* February 1-2, 2012.

[17] Pyakurel, U. and Dhamala, T.N., (2015). Models and algorithms on contraflow evacuation planning network problems. *International Journal of Operations Research*, **12**(2), 36-46.

[18] Pyakurel, U., Dhamala, T.N. and Dempe, S. (2017). Efficient continuous contraflow algorithms for evacuation planning problems. *Annals of Operations Research (ANOR)*, **254**, **335-364**.

[19] Pyakurel, U., Nath, H.N., Dempe, S. and Dhamala, T.N., (2019). Efficient dynamic flow algorithms for evacuation planning problems with partial lane reversal. *Mathematics*, **7**, **1-29**.

[20] Pyakurel, U., Nath, H.N. and Dhamala, T.N. (2018). Efficient contraflow algorithms for quickest evacuation planning, *Science China Mathematics*, **61**, 2079–2100.