Efficient Algorithm for Incrementally Deploying Content Objects Over Networks

Lung-Pin Chen and Jhen-You Hong
Dept. Computer Science and Information Engineering,
Tunghai University, Taichung, Taiwan
{lbchen,g96350040}@thu.edu.tw

Abstract

Deploying and managing content objects efficiently is critical for building a scalable and transparent content delivery system. This paper investigates the advanced incremental version of the content deployment problem of which the objects are delivered in a successive manner. Recently, the researchers show that the minimum-cost content deployment can be obtained by reducing the problem to the well-known network flow problem. In this paper, the maximum flow problem for a single graph is extended to an incremental growing graph. We show that the time complexity of deriving k maximum flow values of incremental graphs $N_1, N_2, \cdots, N_k$ is no more than that of the single graph N_k. Based on this property, this work develops an efficient algorithm for incrementally deploying content objects over networks.

1 Introduction

In a content delivery system, the *dynamic* content object are constructed by running application programs on *base data* which may change frequently. When a new version of an object is generated, this object should be re-deployed to keep the client's data up-to-date. Essentially, a new version of object can be updated via a direct network transmission. However, with this naive approach, the content object is entirely transmitted even only a small change have been made on the previous version. The *transcoding* [8] is an important technique to reduce the cost of deploying objects. A transcoding operation is a client-side computation process to construct an object from the information of its predecessors. The cost of deploying a set of content objects can be reduced by taking the trade-off between direct transimission and client-side transcoding. Recently, the researchers [8] showed that the object deployment problem can be solved by using the existing *network flow* algorithms [3, 4, 6, 5, 7].

Due to the diversity of Internet applications, it would be desirable to provide an efficient way of *incrementally* deploying the content objects between clients and servers over networks. For example, in an e-learning system, usually a user accesses the course content in an incremental manner, according to the learning progress, instead of accessing the entire course content at one shot. In this paper, we study the *incrementl content object deployment problem*. We first introduce the notion of *incremental flow networks*, which is a sequence of incrementally growing flow networks $(N_1, N_2, \cdots, N_k)$. Then, we show that an incremental content object deployment problem can be efficiently solved by using our new incremental maximum flow algorithm.

The rest parts of this paper is organized as follows. In Section 2, we define the basic content deployment problem. Also, the reduction from this problem to the flow network problem is discussed. In Section 3, the efficient incremental maximum flow algorithm and content deployment algorithm are discussed. Finally, conclusions are made in Section 4.

2 Basic Content Deployment Algorithm

2.1 Content Deployment Problem

In this paper, the digital content is modeled as an ODG (*object dependency graph*) $G = (U, F)$, where each $v \in U$ refers to a *content object* (or, simply, *object*); each edge $(u, v) \in F$ refers to the dependency relation from object u to v. To model the transcoding operation, when deploying, all the objects in graph G are classeified into three types:

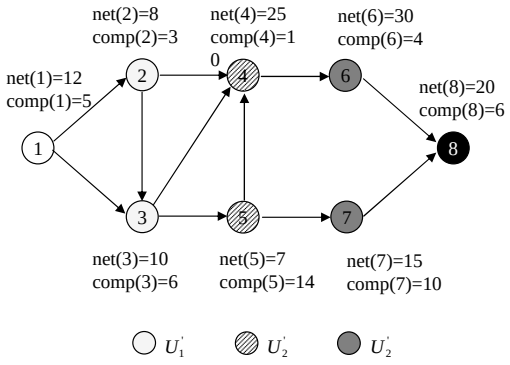

Figure 1: An object dependency graph G and target object sets U_1', U_2', U_3'.

O-node, *N-node*, and *C-node*. An object v is O-node if v is not deployed to the client. Also, an object v is N-node (or C-node) if v is deployed via a direct network transmission (or via the client-side transcoding). The sets of O-nodes, N-nodes, and C-nodes in G are denoted by U_O, U_N, and U_C respectively.

Given a graph $G = (U, F)$ and a set of *target objects* $U' \subseteq U$, a *content object deployment* is a five tuple $\mathcal{D} = (G, U', U_O, U_N, U_C)$ where $\{U_O, U_N, U_C\}$ is a partition of vertices of G such that the following properties hold:

- All the target objects must be deployed to the client, *i.e.* $U' \subseteq \{U_N \cup U_C\}$, and
- If object v is a C-node (constructed via the client-side transcoding) then all of its predecessors must be deployed to the client. That is, for all $v \in U$ and $(u, v) \in F$, $v \in U_C \Rightarrow u \in \{U_N \cup U_C\}$.

Let $net(v)$ denote the cost of transmitting object v from the server to the client, and $comp(v)$ denote the cost of transcoding object v from all of its predecessors. The *cost* of content deployment $\mathcal{D} = (G, U', U_O, U_N, U_C)$ is defined as $cost(\mathcal{D}) = \sum_{v \in U_N} net(v) + \sum_{v \in U_C} comp(v)$. The *minimum content deployment*, with respect to ODG G and target object set U', is the one with the minimum cost among all the feasible content deployments. For example, for the ODG in Figure 2(a), a content object deployment with $\mathcal{D} = (G, \{6, 7\}, U_O = \phi, U_N = \{5\}, U_C = \{1, 2, 3, 4, 6, 7\})$ is the minimum content deployment for for the target objects $\{6, 7\}$.

2.2 Content Deployment Algorithm

Recently, the researchers [8] shown that a minimum content deployment problem can be solved by reducing the problem to the minimum cut of a flow network N. A *flow network* $N = (V, E, S, T, cap)$ is a five tuple in which (V, E) is a directed graph where each edge $(u, v) \in E$ is associated with the non-negative *capacity* $cap(u, v)$. In the flow network, $S \subset V$ are designated as *source vertices* and $T \subset V$ are designated as *sink vertices*. A source (or sink) vertex only has outgoing (or incoming) edges.

A flow f of N is a function from edges to non-negative values which represents the units of flow been sent from the sources to the sinks without exceeding the edge capacities. That is, the following properties are satisfied: (1) $f(u, v) \leq cap(u, v)$, and (2) $IN(u) = OUT(u)$ for each vertex $u \in V \setminus \{S, T\}$ where $IN(u) = \sum_{v,(v,u) \in E} f(v, u)$ and $OUT(u) = \sum_{v,(u,v) \in E} f(u, v)$.

A *cut* of flow network N is a partition (X, Y) on the vertices of N such that all the source nodes are in X-part and all the sink nodes are in Y-part. The cost a cut $\mathcal{C}$ is $cost(\mathcal{C}) = \{cap(u, v) \mid u \in X, v \in Y\}$. Note that the only the edges from X to Y contribute their capacities to the cut cost (*excluding* those edges from Y to X). Ford and Fulkerson prove the following *max-flow-min-cut theorem* [3].

Theorem 2.1 *Given a flow network, its maximum flow value and minimum cut cost are equal.*

■

The reduction of a content object deployment problem to the flow network problem is illustrated in Algorithm 1 and explained in Algorithm 1.

A reduction from G to N for Step 1 of Algorithm 1 is illustrated in Figure 2. For the target set $U_3' = \{6, 7\}$ in Figure 1, the minimum content object deployment is $\mathcal{D} = (G, \{6, 7\}, U_O = \phi, U_N = \{5\}, U_C = \{1, 2, 3, 4, 6, 7\})$. Observing Figure 1 and Figure 2, we can find that the minimum cut cost of the reduced network N is equal to $cap(5b, 5e) + cap(s, 1b) + cap(s, 2) + cap(s, 3) + cap(s, 4) + cap(s, 6) + cap(s, 7) = net(a) + comp(b)$, $a \in U_N$ and $b \in U_C$. Clearly, this cost is equal to $cost(\mathcal{D})$.

For convinient, a cut with cost not equal to ∞ is caled a *feasible cut*. Also, given a cut $\mathcal{C}$, let $\mathcal{D} = Map(\mathcal{C})$ be the content deployment constructed in Step 3 in Algorithm 1. The Map function establishes the one-to-one mapping between

Algorithm 1 BASIC_CONTENT_DEPLOYMENT (G, U')

1. (**Reduction**) Perform the following vertex-to-edge transformation from ODG $G = (U, F,)$ to flow network $N = (V, E, S, T, cap)$:
   - For each vertex a in the graph G, add two nodes a_{begin} and a_{end} to N. Then, add edge (a_{begin}, a_{end}) and (a_{end}, a_{begin}) to N. Let $cap(a_{begin}, a_{end}) = net(a)$ and $cap(a_{begin}, a_{end}) = \infty$.
   - Add a source node s to N. Then, add edge (s, a_{begin}) to N and let $cap(s, a_{begin}) = comp(a)$ for all objects a.
   - If an edge (a, b) exists in G then add an ∞-capacity edge (a_{end}, b_{begin}) in N.
   - For all target objects a in U', set a_{end} as a sink node of N.
2. After the flow network N is constructed, find the minimum cut $\mathcal{C} = (X, Y)$ of N by using the existing maximum flow algorithms [8].
3. (**Mapping**) Construct the minimum deployment $\mathcal{D} = (G, U', U_O, U_N, U_C)$ based on the following mapping:
   - Object $a \in U_O \Leftrightarrow a_{begin} \in X$ and $a_{end} \in X$ in N
   - Object $a \in U_N \Leftrightarrow a_{begin} \in X$ and $a_{end} \in Y$ in N
   - Object $a \in U_C \Leftrightarrow a_{begin} \in Y$ and $a_{end} \in Y$ in N
4. return $\mathcal{D}$

cuts of N and deployments of G. This mapping confirms the correctness of Algorithm 1 as proved in Lemma 2.1.

Lemma 2.1 *For an ODG $G = (U, F)$ and a set of target objects $U' \subset U$. Let N be the flow network constructed in Step 1 in Algorithm 1. The Map function is an one-to-one mapping between feasible cuts of N and content deployments of G. Also, $cost(\mathcal{C}) = cost(Map(\mathcal{C}))$.*

Proof. Note that the proof is simplified from that in [8]. Since the non-feasible cut with cost ∞ can not be a minimum cut, it is enough to consider only feasible cuts. For each feasible cut

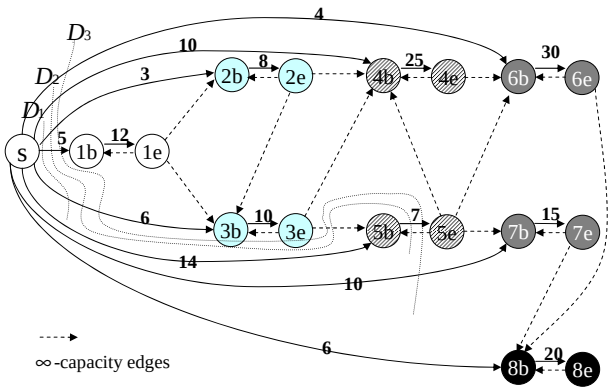

Figure 2: An object dependency graph G and target object sets U'_1, U'_2, U'_3.

$\mathcal{C} = (X, Y)$ of N, assume that $\mathcal{D} = Map(\mathcal{C}) = (G, U', U_O, U_X, U_C)$ is constructed in Step 3 in the algorithm.

First, we show that for each $\mathcal{C}$, the corresponding $Map(\mathcal{C})$ is a valid content deployment of G. Recall that N has nodes $\{s\} \cup \{a_{begin}, a_{end} \mid a \text{ is an object in } G\}$. In cut $\mathcal{C} = (X, Y)$, if a node $b_{begin} \in Y$ then $a_{end} \in Y$ for all a_{end} with $cap(a_{end}, b_{begin}) = \infty$. (Otherwise, $\mathcal{C}$ has cost ∞ and can not be a feasible cut.) From the mapping rules in Step 3 in the algorithm, we can derive that in $\mathcal{D}$,

$$b \in U_C \implies a \in \{U_N \cup U_C\}, \forall (a, b) \text{ in } G$$

Also, since $a_{end} \in Y$ for all taget objects a. This impies that in $\mathcal{D}$,

$$a \in \{U_N \cup U_C\}, \forall a \in U'$$

From above, $\mathcal{D}$ is a valid deployment of G and its cost is equal to that of cut $\mathcal{C}$.■

3 Incremental Content Deployment Algorithms

3.1 Incremental Minimum Content Deployment Problem

For two vertices u and v in an ODG G, denote it by $u \preceq v$ if there is a path from u to v. Moreover, for two vertex sets S_1 and S_2, $S_1 \preceq S_2$ if and only if $u \preceq v$ for each $u \in S_1$ and $v \in S_2$. A sequence of content deployments is called the *incremental content deployments* if all of them are defined on an same ODG but with incrementally growing target object sets. Formally,

let $\mathcal{D}_i = (G, U'_i, U_{O_i}, U_{N_i}, U_{C_i}), i = 1, 2, \cdots, k$, then, $(\mathcal{D}_1, D_2, \cdots, D_k)$ is an incremental content deployment if $U'_1 \preceq U'_2 \preceq \cdots \preceq U'_k$. Furthermore, $(\mathcal{D}_1, D_2, \cdots, D_k)$ is said to be minimum if each member $\mathcal{D}_i$ is minimum with respect to the target set U'_i.

3.2 Abstraction of Network Flow Algorithms

For example, consider the target set $U'_1 = \{2, 3\}$, $U'_2 = \{4, 5\}$, and $U'_3 = \{6, 7\}$ in Figure 1. Since $U'_1 \preceq U'_2 \preceq U'_3$, these target sets define an incremental content deployment problem. The minimum deploment for U'_1, U'_2, and U'_3 are labeled as D_1, D_2, and D_3 in Figure 2, respectively.

In this subsection, we provide a high-level abstraction for the flow network algorithms. The abstraction is helpful on extending the basic content deploying algorithm to the incremental version.

3.2.1 Generic Preflow Algorithm

A *preflow* f on a flow network is a flow except that the total flow into a non-source non-sink vertex u can exceed that out of u. Namely, for all $u \in V \setminus \{S \cup T\}$, $IN(u) \geq OUT(u)$ [6]. The difference between the total flow into and out of u is called the *excess flow* and denoted by $ex(u) = IN(u) - OUT(u)$. A non-source non-sink vertex u is called *active* if $ex(u) > 0$. Also, an edge (u, v) with $f(u, v) < cap(u, v)$ is called a *residual edge*. Clearly, there is still at most $cap(u, v) - f(u, v)$ units of flow can be sent via edge (u, v) without violating the capacity constraint. A path with all residual edges is called a *residual path*. Herein after, let n and m denote the number of vertices and edges of N, repsectively. Ford and Fulketson [3] proved Theorem 3.1.

Theorem 3.1 ([3]) *A preflow f is a maximum flow of network N if and only if there is no residual path from sources to sinks on N.*

The *generic preflow algorithm* [6] is a class of maximum flow algorithms that maintain a preflow and work by repeatedly choosing active vertices and sending excess flows along residual paths towards the sinks. Until no residual path can be found in the flow network, based on Theorem 3.1, the preflow becomes a maximum flow.

In order to maintain the preflow efficiently, a *labeling function* d is used to estimate how close the vertices are to the sinks. For a network $N = (V, E, S, T, cap)$, a *valid* labeling function d is a $V \to \mathbb{Z}$ function, such that $d(v) \leq d(w) + 1$ for each residual edge (v, w). Also, $d(s) = n$ for each source $s \in S$, $d(t) = 0$ for each sink $t \in T$.

The algorithm basicly consists of a main loop which repeatedly applies Push/Relabel operations until the maximum flow is obtained. The algorithm is described as follows:

Algorithm 2 GENERIC_PREFLOW $(G, U'_1, U'_2, \cdots, U'_k)$

1. (INIT) Clear all the flow values and labels to 0. For all $s \in S$, send $cap(s, u)$ units of flow from s to u for all $(s, u) \in E$. Set $d(s) = n$ and $d(v) = 0$ for all sources s and non-sources v.

2. Repeatedly select an active vertex u until no active vertex exists:

   - (PUSHRELABEL) Choose and apply one the following applicable operation on u:

     (a) PUSH(v, w)
     Applicable: If v is active and some adjacent edge (v, w) is a residual edge.
     Action: Send $min(cap(v, w) - f(v, w), ex(v))$ units of flow from v to w.

     (b) RELABEL(v)
     Applicable: If v is active and v has *no* adjacent residual edge.
     Action: Increase $d(v)$ by one.

Figure 3 illustrates a valid labeling function on the network. The height of vertex indicates the label value of the vertex. Initially, every non-sink vertice is placed on height 0. A vertex v (*e.g.* node 3, 5 or 7) is lifted if it is still active but no residual edge (v, w) with $d(v) = d(w) + 1$ can be found.

3.2.2 Abstraction of Generic Preflow Algoroithm

After performing an operation, the *state* of the related data structures of the generic preflow algorithm is changed, as defined as follows.

Definition 3.1 *A* state *q with repect to a flow network N is a three tuple $q = (N, f, d)$, where f is a preflow and d is a labeling function. Two states $q = (N, f, d)$ and $q' = (N', f', d')$ are said to*

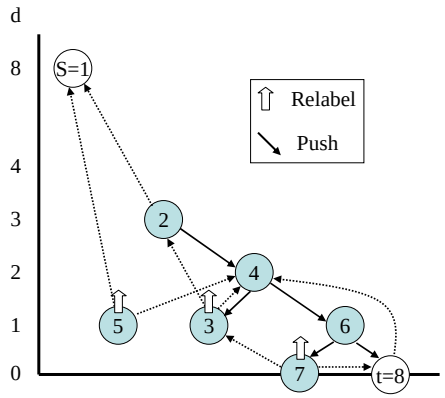

Figure 3: Illustration of Push/Relabel operations. These operations can be applied in an arbitrary order. Note that all edges depicted are residual edges.

be equal *if* $N = N'$, $f = f'$ *and* $d = d'$ *except that the source/sink nodes in* N *and* N' *can be different.* ■

The execution of function PushRelabel of generic preflow algorithm can be represented as a sequence

$$\mathcal{I} = (q^0, (op^1, q^1), (op^2, q^2), \cdots, (op^L, q^L)) \quad (1)$$

, where q^i is the state immediately after applying i-th operation op^i, and L is the total number of operations performed. The first q^0 and last q^L refer to the state before and after executing all the L operations. Hereinafter, such sequence $\mathcal{I}$ is called an *instance* of generic preflow algorithm, and the set of all instances is denoted by Π. An instance $\mathcal{I} = (q^0, (op^1, q^1), (op^2, q^2), \cdots, (op^L, q^L))$ is called a *prefix* of another instance $\mathcal{I}' = (q'^0, (op'^1, q'^1), (op'^2, q'^2), \cdots, (op'x^{L'}, q'^{L'}))$ if $q^j = q'^j$ and $op^j = op'^j$ for all j, $0 \leq j \leq L$.

The generic preflow algorithm provides us with a very useful guideline: Any instance $\mathcal{I} \in \Pi$ can find the maximum flow since Push and Relabel operations can be applied in an *arbitrary* order.

Lemma 3.1 *Assume that* $\mathcal{I}$ *is an instance of the generic preflow algorithm for the input flow network* N*. If the first state* q^0 *is a valid state and there is no active vertices in the last state* q^L*, then the execution of instance* $\mathcal{I}$ *obtains the correct maximum flow value of* N*. Additionally, the length of any instance* $\mathcal{I}$ *is* $O(n^2m)$*, where* n *and* m *is the number of nodes and edges of the flow network* N*, respectively.*

Proof. The properties are discussed in [6] and are omitted in this paper.■

3.3 Incremental Maximum Flow Algorithm

A sequence of flow networks $(N_1, N_2, \cdots, N_k)$ is called a set of *incremental flow networks* if N_i and N_{i+1} are the same execpt that some sink nodes in N_i turn to non-sink in N_{i+1}. Formally, assume that $N_i = (V, E, S, T_i, cap)$, then, $(N_1, N_2, \cdots, N_k)$ is incremental if and ony if $T_{i+1} \subseteq T_i$ for each i, $1 \leq i < k$.

The incremental maximum flow alorithm comprises of a sequence of basic maximum flow functions, as described in Algorithm 3.

Algorithm 3 Incremental_Preflow $(G, V_1', V_2', \cdots, V_k')$

1: Construct $N = (V, E, S, T, cap)$ from graph G.
2: Let state $q_1^0 = \{N_1, f, d\}$ be the state right after invoking function Init (in Algorithm 2)
3: **for** stage $i = 1$ to k **do**
4: Let $N_i = (V, E, S, T_i, cap)$ with $T_i = V_i'$.
5: **if** $i > 1$ **then**
6: Let $q_i^0 = q_{i-1}^{L_{i-1}}$ but with different sink V_i' *//see Step 4 and Definition 3.1*
7: **end if**
8: Invoke function PushRelabel() on state q_i^1. Let $q_i^{L_i} = \{N_i, f, d\}$ be the last state after performing L_i operations in the function.
9: **end for**
10: **return** the maximum flow values of the last states $q_i^{L_i}$ of all stages $i = 1, 2, \cdots, k$.

Assume that in Algorithm 3, L_i is the number of the operations performed by PushRelabel() in stage i. The sequence of operations performed in stage i is represented by $\mathcal{I}_i = (q_i^0, (op_i^1, q_i^1), (op_i^2, q_i^2), \cdots, (op_i^{L_i}, q_i^{L_i}))$, which is similiar to Equation 1 but plus an additional subscript i.

The correctness and time complexity of Algorithm Incremental_Preflow are illustrated in Figure 4 and discussed in Lemma 3.2.

Lemma 3.2 *At the end of each stage* i *of Algorithm* Incremental_Preflow*, the maximum flow value of* N_i *can be obtained from the last state* $q_i^{L_i}$*. Also, the total time complexity of Algorithm* Incremental_Preflow*, from stages* 1 *to* k*, is* $O(n^2m)$*, where* n *and* m *is the number of vertices and edges of flow network* N_k*, respectively.*

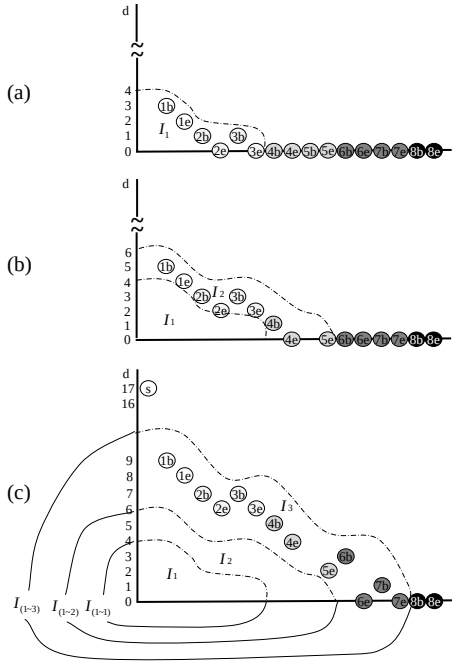

Figure 4: The sequence of operations performed in stage $1, 2, \cdots, i$ of the incremental flow network.

Proof. In Algorithm 3, each stage i uses the last state in previous stage $i-1$ as its first state, *i.e.* $q_i^0 = q_{i-1}^{L_{i-1}}$ but change N_{i-1} to N_i. The proof is illustrated in Figure4. Next, we show that q_i^0 is a valid state for stage i. Recall that a state is valid if the condition $d(v) \leq d(w) + 1$ holds for every residual edge (v, w). If $q_{i-1}^{L_{i-1}} = \{N_{i-1}, f, d\}$ is valid for stage $i-1$. Since $N_i = N_{i-1}$ except that some sink in N_{i-1} changes to non-sink in N_i. This implies that $q_i^0 = \{N_i, f, d\}$ is a valid state for N_i.

For convinient, let $\mathcal{I}_{(1\sim i)}$ be the concatenience of the operation sequences from stage 1 to i, that is,

$$\begin{aligned}
\mathcal{I}_{(1\sim 1)} &= (\mathcal{I}_1) \\
\mathcal{I}_{(1\sim 2)} &= (\mathcal{I}_1, \mathcal{I}_2) \\
\mathcal{I}_{(1\sim 3)} &= (\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3) \\
&\cdots \\
\mathcal{I}_{(1\sim k)} &= (\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_k)
\end{aligned}$$

From above, each $\mathcal{I}_{(1\sim i)}$ is an instance for the flow network N_i and can derive the maximum flow value of N_i correctly. Furthermore, since $\mathcal{I}_{(1\sim i)}$ is the prefix of $\mathcal{I}_{(1\sim i+1)}$, the total time to perform all the instances $\mathcal{I}_{(1\sim 1)}, \mathcal{I}_{(1\sim 2)}, \cdots, \mathcal{I}_{(1\sim k)}$ is only $O(n^2 m)$ according to Lemma 3.1.■

3.4 Incremental Content Deployment Algorithm

This subsection shows that the incremental minimum content deployment can be derived by reducing the problem to the incremental flow network problem. Lemma 3.3 proves this property.

Lemma 3.3 *Consider a sequence of incremental target object sets $(U_1', U_2', \cdots, U_k')$ of ODG $G = (U, F)$. Let $(R(G, U_1'), R(G, U_2'), \cdots, R(G, U_k')$ be the reduced flow networks in Step 1 in Aglorithm 1. Then, $(R(G, U_1'), R(G, U_2'), \cdots, R(G, U_k')$ is a set of incremental flow networks, and the maximum flow value of $R(U_i')$ is equal to the cost of minimum content deployment of G with target object set U_i', for each $i = 1, 2, \cdots, k$.*

Proof. Assume that $N_i = R(U_i') = (V, E, S, T_i, cap)$, $1 \leq i \leq k$. Since $U_1' \preceq U_2' \preceq \cdots \preceq U_k'$, the transformed flow networks $(R(G, U_1'), R(G, U_2'), \cdots, R(G, U_k'))$ are all the same but with different sink nodes and the following properties can be derived:

$$\begin{aligned}
& U_i' \preceq U_{i+1}' \\
\Rightarrow \quad & pred(U_i') \preceq pred(U_{i+1}') \\
\Rightarrow \quad & V \setminus pred(U_{i+1}') \subset V \setminus pred(U_i') \\
\Rightarrow \quad & T_{i+1} \subseteq T_i
\end{aligned}$$

, where $pred(S)$ is the set of vertices u with path to some node in S, and $V \setminus pred(S)$ refers to the set of vertices with path from nodes in S. Recall that in Algorithm 1, $V \setminus pred(U_i')$ is set as the sink nodes.

From above, $(R(G, U_1'), R(G, U_2'), \cdots, R(G, U_k'))$ are incremental flow networks. Furthermore, from Lemma 2.1, the maximum flow value of network $R(G, U_i')$ is equal to the cost of minimum deployment of ODG G with respect to target object set U_i'. Thus, the lemma follows.■

4 Discussion

In this paper, the maximum flow problem for a single graph is extended to an incremental growing graph. We show that the time complexity of deriving k maximum flow values of incremental graphs $N_1, N_2, \cdots, N_k$ is no more than that of the single graph N_k. The extension can be used to solve the incremental content object deployment problem, which is shown can be solved by reducing the problem to the flow network problem.

ACKNOWLEDGMENTS

The authors would like to thank the National Science Council of the Republic of China for financially supporting this research under contract No NSC 97-2221-E-029 -022

References

[1] L. Bouge. Repeated snapshots in distributed systems with synchronous communication and their implementation in CSP. *Theoretical Comput. Sci*, 49:145–169, 1987.

[2] L.B. Chen and I.C. Wu. On the time complexity of minimum and maximum global snapshot problems. *Information Processing Letters*, 67:151–156, 1998.

[3] L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Can. J. Math.*, 8:399–404, 1956.

[4] L.R. Ford and D.R. Fulkerson. *Flows in Networks*. Princeton Univ. Press, Princeton, NJ, 1962.

[5] A.V. Goldberg. *Recent Developments in Maximum Flow Problems*. Technical report 98-045, NEC Research Institute, Inc., 1998.

[6] A.V. Goldberg and R.E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, October 1988.

[7] R.K. Ahuja T.L. Magnanti and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[8] Xueyan Tang and Samuel T. Chanson. Minimal cost replication of dynamic web contents under flat update delivery. *IEEE Transactions on Parallel and Distributed Systems*, 15:431–441, May 2004.