Distrib Parallel Databases (2007) 22:1–25
DOI 10.1007/s10619-006-7006-4

Disseminating dependent data in wireless broadcast environments

Chuan-Ming Liu · Kun-Feng Lin

Published online: 3 January 2007

Abstract In wireless mobile environments, data broadcasting is an effective approach to disseminate information to mobile clients. In some applications, the access pattern of all the data can be represented by a weighted DAG. In this paper, we explore how to efficiently generate the broadcast schedule in a wireless environment for the data set having a weighted DAG access pattern. Such a broadcast schedule not only minimizes the access latency but also is a topological ordering of the DAG. Minimized access latency ensures the quality of service (QoS). We prove that it is NP-hard to find an optimal broadcast schedule and provide some heuristics. After giving an analysis for these heuristics on the latency and complexity, we implement all the proposed heuristics to compare their performance.

Keywords Data broadcasting · Latency · Directed acyclic graphs · NP-hard · Access pattern

1 Introduction

Data broadcasting provides an efficient way to disseminate data in the wireless environments and has attracted much research attention in recent years [1, 7, 8, 11, 13–15]. In a wireless environment, a server can provide public information (e.g. electronic news service, traffic information, stock-price information, etc.) to a large client population using broadcasting. Clients can access the information by tuning into the

Recommended by: Sunil Prabhakar

C.-M. Liu (✉) · K.-F. Lin
Department of Computer Science and Information Engineering
National Taipei University of Technology, Taipei 106, Taiwan
e-mail: cmliu@csie.ntut.edu.tw

K.-F. Lin
e-mail: ivan@mcse.csie.ntut.edu.tw

broadcast. One key issue related to data broadcasting is how the server organizes the data in the broadcast in order that the clients can access the information efficiently in terms of the *latency* (time elapsed from requesting to receiving data) and the *tuning time* (the amount of time spent listening to the broadcast). Short latency and tuning time can ensure the quality of service (QoS) provided by the server.

Most of the previous work about the broadcast scheduling assumed that the broadcast data requested are independent [10, 17]. However, in some applications, the data requested are dependent. For example, consider the scenario that a user requests a Web page which contains some components, such as audio clips or images. When a mobile client uses a browser to access the Web page, the browser will request all the components contained in the Web page after receiving the Web page. Such a request leads to an access pattern among the dependent data [2, 9, 11]. The other application can be found when a mobile client has an inquiry about the stock price of some company. It is highly possible that the mobile client would like to know the stock information of some other related companies. In this case, the information among all the related companies is dependently accessed. Besides, consider the on-demand broadcasting model in [16] where the server collects a batch of requests from the mobile clients and then sends the results to that group of clients via broadcast. After receiving a signal from the server, each mobile client starts to receive the result from the beginning of the broadcast cycle. The data broadcast in this model are correlated and dependent.

In order to minimize the latency, many of the previous papers considered the given data access frequency [1, 8, 10, 14, 17] or the data access pattern [2, 9, 11]. With such assumptions, how the server schedules the data is the major challenge. An important issue when designing the broadcast schedule with minimized latency is to consider whether the broadcast allows a mobile client to receive the data in the middle of a broadcast cycle. Allowing the users to receive data in the middle of a cycle can reduce the latency by skipping the waiting time for the beginning of a cycle. However, for the scenarios mentioned in the previous paragraph, it is not proper to allow a client to receive the data in the middle of a cycle due to the data dependency. We thus discuss the case that the mobile client can only start to receive data from the beginning of a broadcast cycle. Such a consideration distinguishes our work from the others [2, 9, 11]. In this paper, we consider the access pattern formed by all the requests where the data dependency and access frequency can be represented by a weighted *directed acyclic graph, DAG* [11]. We investigate how to generate the broadcast schedule in order to minimize the average latency when the broadcast data has the access pattern as a weighted DAG.

According to the data dependency, we consider the generated broadcast schedule is a *topological ordering* of the vertices in the input DAG G. A *topological ordering* of a DAG G is an ordering of the vertices in G where the precedent vertices in the DAG are broadcast earlier than the following vertices. In other words, for an edge (u, v) in a DAG G, vertex u should be broadcast before vertex v in the same cycle. The broadcast schedule generated without considering the weights (access frequencies) and without the topological ordering may result in a longer latency. We show that it is NP-hard to find an optimal broadcast schedule which is a topological ordering of the input DAG and results in a minimum latency.

Due to the computational intractability, we propose several heuristic methods. These heuristics can be classified into two classes by the underlying strategies. One is the *level-oriented strategy* which first considers the level of each vertex and then assigns a position to each vertex. Different kinds of definitions for level will lead to different broadcast schedules. The other is the *greedy strategy* where the broadcast schedule is generated by traversing the input DAG in a greedy fashion. In addition, we further propose a refinement algorithm which can achieve a better performance for the broadcast schedule which is a topological ordering of the vertices in terms of latency.

The organization for the rest of this paper is as follows. In Section 2, we give some related works as well as the terminology and notations used in this paper. We then formulate the problem formally in Section 3. After showing that the proposed problem is NP-complete in Section 4, we present and analyze the algorithms using the level-oriented strategy in Section 5, the algorithms using the greedy strategy in Section 6, and the refinement algorithm in Section 7, respectively. In Section 8, we show the simulation. Section 9 concludes this paper.

2 Preliminary

One objective of data broadcast scheduling is to minimize the access latency. As mentioned, allowing a mobile client to start receiving the data in the middle of a broadcast cycle can reduce the access latency. Thus, many papers considered to minimize the relative distance between the positions of two related data items in order to minimize the access latency [2, 4, 11, 14]. In [2], the authors considered three types of relation between two accessed data items, including *AND*, *OR*, and *IMPLY* relations. They assumed that it takes one unit of time (a *slot*) to broadcast each file. For each kind of relation, they provided a heuristic to generate the broadcast which minimizes the latency. In [14], the authors considered data allocation and query processing on wireless broadcast channels. They assumed that the access pattern can be modeled as a *directed graph*. In their solution, the directed graph was transferred into an access forest by cutting some edges. Each tree in the forest may have an optimal schedule locally. Then the broadcast schedule was generated by combining those optimal schedules of the trees in a global viewpoint. However, the resulting schedule may not be optimal in terms of latency since some edges had been removed when transferring the graph into a forest.

Chehadeh et al. [4] assumed that the relation among the data to be accessed can be represented by a *directed acyclic graph* (DAG) and presented several heuristics according to the following three criteria, (1) linear ordering (i.e topological ordering), (2) minimum linear distance between any two related objects and (3) more availability for popular objects. The authors considered either the case that the generated schedule is a topological ordering for an un-weighted DAG or the case that the resulting schedule is not necessarily a topological ordering for a weighted DAG. In our work, the generated broadcast schedule for a weighted DAG is a topological ordering. We consider the case that each mobile client starts to receive data from the beginning of a broadcast cycle and focus on how to minimize the latency in an overall viewpoint. In the next section, we will give our problem a formal definition.

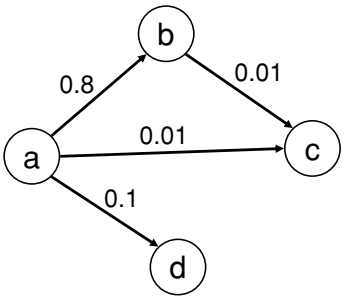


Fig. 1 A weighted DAG of 4 vertices; the weight on an edge (u, v) representing the access probability for vertex v after vertex u has been accessed

A DAG [3] $G = (V, E)$ is a directed graph that has no cycles, where V is a finite set of vertices and E is a binary relation on V. The set E is called the *edge set* of G and consists of ordered pairs of vertices. An element (u, v) in E, $u, v \in V$ and $u \neq v$, is an *edge*. If (u, v) is an edge in G, we say that vertex $u(v)$ is *adjacent* to(from) vertex $v(u)$. We also call the edge (u, v) as the *in-edge* (*out-edge*) of $v(u)$ and the *in-degree* (*out-degree*) of a vertex is the number of edges entering(leaving) it. We define *in-degree* (*out-degree*) $d_i(d_o)$ of a DAG G to be the maximum in-degree (out-degree) among all the vertices in G. The *degree* d of a DAG G is $\max\{d_i, d_o\}$. Among the vertices in V, a *source* is a vertex which has no any in-edge and a *sink* is a vertex having no out-edge. If there is a path $P(u, v)$ from u to v, we say that v is *reachable* from u via $P(u, v)$ and u is a *prior vertex* of v as well as v is a *posterior vertex* of u. For a vertex v, we further define $PRE(v)$ as the set of all the vertices adjacent to v and $ADJ(v)$ as the set of all the vertices adjacent from v.

For each edge (u, v) in the input DAG, we consider that there is a weight $p_{uv} < 1$ on that edge. For any edge (u, v), the direction from vertex u to vertex v represents that u is accessed before v and the weight $p_{uv} \leq 1$ on (u, v) stands for the conditional probability of accessing vertices v after vertex u has been accessed. As shown in Fig. 1, after vertex a has been accessed, the chances to access vertex b, c, and d are $p_{ab} = 0.8$, $p_{ac} = 0.01$, and $p_{ad} = 0.1$, respectively. The summation of all the weights on the edges from a vertex u is less than or equal to 1.

3 Problems

Suppose that the access pattern of the data to be broadcasted is modeled as an edge-weighted DAG $G = (V, E)$. We assume that there is only one source s in the DAG. If the input DAG has multiple sources (no in-edge to them), we add a pseudo-vertex s to the DAG which has a directed edge to each source with equal weight respectively.

For each vertex v except the source s in the DAG G, there is at least one path from s to vertex v. Suppose $P(s, v)$ is one of the paths from s to v. We define the probability to access vertex v via path $P(s, v)$ as $\prod_{(i,j)\in P(s,v)} p_{ij}$. Let $\mathbb{P}_v$ be the set of all the paths from the source s to vertex v. The probability to access vertex v from the source s is

$$\gamma(v) = \sum_{P(s,v)\in\mathbb{P}_v} \left(\prod_{(i,j)\in P(s,v)} p_{ij} \right). \tag{1}$$

For example, consider the DAG in Fig. 1. The $\gamma(c) = 0.8 \times 0.01 + 0.01 = 0.018$.

Suppose that a generated broadcast schedule f is a 1-to-1 mapping from V to the set $\{1, \ldots, |V|\}$. Then, the position of vertex v in the broadcasting channel is $f(v)$. The expected latency for accessing vertex v via the broadcast schedule f is therefore $f(v) \cdot \gamma(v)$. Hence the *overall latency* for accessing all the data can be

$$\sum_{v \in V} f(v) \cdot \gamma(v). \tag{2}$$

This paper discusses the problem about how to generate a broadcast schedule which (1) is a topological ordering of the vertices in the input DAG and (2) has the minimum overall latency as in Eq. (2). We refer to such a problem as the *Topological Ordering Broadcast with Minimum Latency (TOBML) Problem* and give a formal definition for it in Definition 1. In the following sections, we will show that the TOBML problem is NP-complete and provide some heuristics for it.

Definition 1 (TOBML). Suppose all the notations are defined as above. The Topological Ordering Broadcast with Minimum Latency Problem is to find a 1-to-1 mapping $f : V \to \{1, 2, \ldots, |V|\}$ such that

1. if $(u, v) \in E$, then $f(u) < f(v)$, and
2. $\sum_{v \in V} f(v) \cdot \gamma(v)$ is minimized.

4 NP-completeness

To show that the TOBML problem is NP-Complete, we consider the decision problem, *Topological Ordering Broadcast with a Given Latency (TOBGL) Problem*, defined as below.

Definition 2 (TOBGL).
Instance: A directed acyclic graph (DAG) $G = (V, E)$, an integer K, and a weight $p_{uv} \leq 1$ for each edge $(u, v) \in E$.
Question: Does there exist a 1-to-1 mapping $f : V \to \{1, 2, \ldots, |V|\}$ such that

1. if $(u, v) \in E$, then $f(u) < f(v)$, and
2. $\sum_{v \in V} f(v) \cdot \gamma(v) \leq K$.

Clearly, if the TOBGL problem is NP-complete, then TOBML is also NP-complete. We therefore focus on showing TOBGL is NP-complete. Consider a special type of DAG defined in Definition 3 for the TOBGL problem.

Definition 3 (Equally Out-edge Weight DAG). An *Equally Out-edge Weight DAG* is a DAG in which (1) all the edges from a vertex have the same weight and (2) the sum of all the weights on the edges from a vertex is equal to 1.

When the input is an equally out-edge weight DAG for the TOBGL problem, we refer to such a problem as the ETOBGL problem. We then claim that the ETOBGL

problem is NP-complete. ETOBGL is in NP since, given a 1-to-1 mapping f, it takes linear time to determine whether $f(u) < f(v)$ if $(u, v) \in E$ and whether $\sum_{v \in V} f(v) \cdot \gamma(v) \leq K$. To show that ETOBGL is NP-hard, we present a reduction from the *Minimum Linear Arrangement of Directed Graph (MINLARD) Problem* defined in Definition 4 which is NP-hard [6].

Definition 4 (MINLARD).
Instance: A directed acyclic graph (DAG) $G' = (V', E')$ and an integer K'.
Question: Does there exist a 1-to-1 mapping $g : V' \rightarrow \{1, 2, \cdots, |V'|\}$ such that

1. if $(u, v) \in E'$, then $g(u) < g(v)$, and
2. $\sum_{(u,v) \in V'} [g(v) - g(u)] \leq K'$.

In the following, we show the reduction. Suppose I' is an instance of the MINLARD problem. An instance I of the ETOBGL problem can be constructed from I' as follows.

1. Let $V = V'$ and $E = E'$.
2. For each v in V, all the out-edges of v have the same weight, $\frac{1}{d_v}$ which is the inverse of v's out-degree d_v.
3. The value of K can be derived by the following relation between K and K'. Suppose the 1-to-1 mapping g exists in I'. We then consider $K = \sum_{v \in V} \gamma(v) \cdot f(v)$. Note that g is a permutation of f and for each vertex v in V, there exists a vertex u in V such that $f(v) = g(u)$. Then we can get

$$K = \sum_{v \in V} \gamma(v) \cdot f(v) = \sum_{v \in V'} \gamma(v) \cdot g(u)$$
$$= \sum_{v \in V'} \gamma(v) \cdot \{[g(u) - g(a_1)] + [g(a_1) - g(a_2)] + \cdots + [g(a_l) - g(s)] + g(s)\},$$

where $a_1, a_2, \ldots, a_l$ are the vertices along the longest path from s to u. Since $g(s) = 1$,

$$K \leq \sum_{v \in V'} \gamma(v) \cdot \left\{ \left\{ \sum_{(i,j) \in E'} g(j) - g(i) \right\} + 1 \right\}.$$

By the property of the 1-to-1 mapping g,

$$K \leq \sum_{v \in V'} \gamma(v) \cdot \{K' + 1\} = \{K' + 1\} \sum_{v \in V'} \gamma(v),$$

where $\sum_{v \in V'} \gamma(v)$ is a known constant.

Then, given an arbitrary instance I' of the MINLARD problem, we can construct an instance I of the ETOBGL problem and such a construction can be done in O(m) time where m is the number of edges in the input DAG. It then remains to show that there is a solution for an instance I' of the MINLARD problem if and only if there is a solution for the instance I of the ETOBGL problem. However, this can be done

easily by following the relation between K and K'. Hence, we can have the following conclusion:

Theorem 1. *The ETOBGL problem is NP-complete; therefore, the TOBML problem is NP-complete.*

5 Level-oriented heuristics

In the previous section, we have shown that the TOMBL problem is NP-complete. In this and the following sections, we introduce some heuristics for generating the broadcast schedules. The heuristics can be classified into two classes according to the underlying strategies. One is the *level-oriented* strategy which first considers the level of each vertex and then assigns the position to each vertex. The other strategy generates the broadcast schedule by traversing the input DAG in a *greedy* fashion and will be discussed in Section 6.

In the level-oriented strategy, the *level* of each vertex confines the placement of a vertex in the broadcast and can make all the vertices in a topological ordering in the broadcast. We provide two heuristics based on the level-oriented strategy. These two heuristics differ in how the level of a vertex is defined. For a vertex v, the *forward-level* of v is the number of vertices on the longest path from the source to v itself. On the other hand, the *backward-level* of v is defined on the contrary and is the number of vertices on the shortest path from v to a sink. After every vertex has its level, the level-oriented strategy moves to the next step and considers the vertices level by level. For the vertices on the same level, the vertex with larger γ value is placed at an earlier position in the broadcast. We next give more details on these two heuristics and then analyze them.

5.1 Forward minimum offset algorithm

We first introduce the *Forward Minimum Offset Algorithm (FMOA)* which uses the forward-level as the level of each vertex. The basic idea of FMOA is as follows. Suppose f^* and f are the best broadcast schedule and a generated schedule, respectively, and each of both schedules is a topological ordering of the vertices. The objective is to minimize $\sum_{v \in V} |f(v) - f^*(v)|$. To achieve this, we first consider the forward-level of each vertex instead of the exact position of each vertex in the broadcast. Note that the levels of the vertices confine the placement for the vertices and implicitly present a topological ordering among the vertices. Furthermore, the forward-level of a vertex v is the best possible position for v to be placed according to the topological ordering. A vertex v having a smaller level should be broadcast earlier. After having the forward-level of each vertex, we then assign each vertex a position in the broadcast.

There are therefore two major steps in *FMOA*: (1) *Assigning Level* step and (2) *Positioning with Minimum Offset* step. Assigning Level step assigns each vertex a forward-level. The source s must be on forward-level 1 according to the topological ordering of the vertices. All the vertices adjacent from s therefore should be on forward-level 2. Process continues based on the following rule: *suppose vertex u is adjacent to vertex v, the forward-level of u must be smaller than the forward-level*

Input: A weighted DAG, $G = (V, E)$ with source s.
Output: A topological-ordering broadcast schedule with minimized latency.

/* Assigning level step */
(1) s is in the queue Q and assigned with level 0, $level(s) = 1$;
each vertex u adjacent from s is assigned with $level(u) = 2$;
$V' = V - \{s\}$;
(2) **while** V' is not empty **do**
(2.1) Pop vertex s from Q;
(2.2) **for** each vertex u adjacent from s **do**
(2.2.1) **if** $level(s) + 1 > level(u)$ **then**
u is assigned with the level $level(u) = level(s) + 1$;
end if
(2.2.2) Decrease the *in-degree* of u by one;
(2.2.3) **if** the *in-degree* of u is zero **then**
Insert u into Q and delete it from V';
end if
(2.2.4) Derive the $\gamma(u)$ by $\gamma(u) = \gamma(u) + \gamma(s) \cdot p_{su}$;
end for
end while
/* Positioning with minimum offset step */
(3) **for** each level l (from forward-level 1 to higher forward-level) **do**
Vertex with larger γ value is placed earlier in the broadcast
(breaking tie by randomly selecting a vertex);
end for

Fig. 2 Forward minimum offset algorithm

of v. When every vertex has been assigned to a forward-level, we move on to the next major step. Positioning with Minimum Offset step places a vertex at a position in the broadcast based on the forward-level of a vertex. A vertex having a smaller forward-level will be places at an earlier position. If two or more vertices have the same forward-level, we use the γ value to decide the position. A vertex with larger γ value will be placed at an earlier position. After these two steps, we then have a broadcast schedule which minimizes the latency and is a topological ordering of the vertices. Figure 2 shows the high-level description of the FMOA algorithm.

We now use the DAG in Fig. 3 to illustrate how FMOA generates the broadcast schedule. In the Assigning Level step, the forward-level of vertex a is 1. Vertices b, c, d, e and g are adjacent from a and form the set $ADJ(a)$. Then, the algorithm assigns forward-level 2 to vertices b, d, c, e and g. After all the vertices in $ADJ(a)$ have been assigned, the algorithm then continues to traverse the vertices in $ADJ(a)$. By removing vertex a and all the edges incident from a, we can have the resulting DAG as shown in Fig. 4.

In $ADJ(a)$, vertices c and d now become the sources. For simplicity and convenience, we store the sources into a queue Q. The algorithm then can traverse vertex c or d. Suppose vertex d is extracted to be traversed next. When vertex d is traversed, since vertex d is assigned to forward-level 2, the vertices b, e, and f adjacent from d are assigned to forward-level 3. Consider vertices b and e which have been assigned

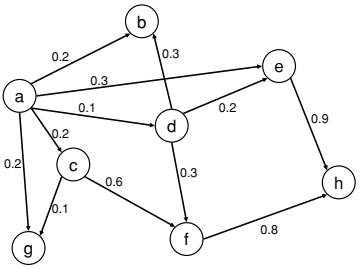


Fig. 3 A weighted DAG where vertex a is a source and b, g, and h are the sinks

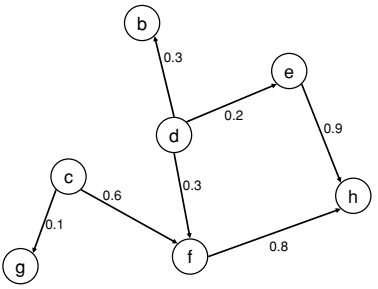
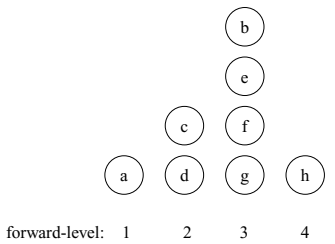


Fig. 4 The DAG after deleting vertex a and all the edges incident from a from the DAG in Fig. 3

to forward-level 2 before. To keep the topological ordering, vertices b and e now should be reassigned to forward-level 3. After removing vertex d and all the edges incident from d, vertices b and e will be the new sources and enqueued into Q. Process continues. Finally, we assign each vertex a forward-level as shown in Fig. 5(a) and can move on to the next major step.

Positioning with Minimum Offset step will decide the final position of each vertex in the broadcast. The γ value of each vertex can be calculated during the Assigning Level step. The position depends first on the forward-level. If two or more vertices have the same level, we then consider the γ value. For example, in Fig. 5(a), vertices c and d are on the same level. Since $\gamma(c) > \gamma(d)$, vertex c should be placed at an



(a) The forward-level assignment for the vertices.

vertex i	a	b	c	d	e	f	g	h
$\gamma(i)$	1	0.23	0.2	0.1	0.32	0.15	0.22	0.408
position	1	5	2	3	4	7	6	8

(b) The corresponding γ value and the resulting position for each vertex in the broadcast.

Fig. 5 Illustration of algorithm FMOA with the DAG in Fig. 3

earlier position than vertex d. The resulting broadcast schedule for the DAG in Fig. 3 is shown in Fig. 5(b).

5.2 Backward maximum offset algorithm

Based on the level-oriented strategy, the *Backward Maximum Offset Algorithm* (*BMOA*) uses the *backward-level* of a vertex instead. The backward-level of a vertex implies that the worst possible position in broadcast schedule for that vertex. Suppose g^* and g are the worst broadcast schedule and a generated schedule, respectively. The objective now becomes to maximize $\sum_{v \in V} |g(v) - g^*(v)|$. A vertex having a larger backward-level therefore should be broadcast earlier. It is similar to FMOA that the backward-levels of the vertices implicitly present a topological ordering among the vertices.

The backward-level of each vertex in BMOA can be obtained by first reversing all the edges of the input DAG and then calculating the backward-level of each vertex in the direction opposite to the direction considered in FMOA. BMOA hence has three major steps: (1) *DAG Reversal* step (2) *Assigning Backward-Level* step and (3) *Positioning with Maximum Offset* step. DAG Reversal step reverses the direction of each edge in the input DAG. Assigning Backward-Level step assigns each vertex's backward-level. For simplicity, we assign each source in the reversed DAG backward-level 1. Process continues based on the following rule: *if vertex u is adjacent from vertex v in the reversed DAG, then the backward-level of u must be larger than the backward-level of v*. When every vertex has been assigned a backward-level, we move on to the next major step. Positioning with Maximum Offset step places a vertex at a position in the broadcast schedule based on the backward-level of a vertex. A vertex having a larger backward-level will be placed at an earlier position. If two or more vertices have the same backward-level, we also use the γ value to decide the position. A vertex with larger γ value will be placed at an earlier position. After these three steps, we then have a broadcast schedule which minimizes the latency. Figure 6 shows algorithm BMOA in details.

Suppose that we apply BMOA on the DAG in Fig. 3. After DAG Reversal step, we can obtain the reversed DAG as shown in Fig. 7. Afterward, the process of BMOA is similar to FMOA. The results for applying BMOA on the DAG in Fig. 3 are shown in Fig. 8.

5.3 Time complexity

It is not difficult to derive the time complexity for FMOA and BMOA. For FMOA, the number of times to traverse a vertex v depends on the in-degree of v and the running time spent on each vertex is constant. If the maximum in-degree among all the vertices of the input DAG is d_i, the time complexity of FMOA is $O(d_i n)$ where n is the number of vertices in the DAG. In BMOA, it takes $O(d_i n)$ time to reverse a DAG. After the reversal, BMOA uses the reversed DAG to assign each vertex to a backward-level and then places the vertices to positions in the same way as FMOA. The time complexity of BMOA is hence $O(\max\{d_i, d_o\}n)$, where d_o is the maximum

Input: A weighted DAG, $G = (V, E)$ with source s.
Output: A topological ordering broadcast schedule with minimized latency.

/* DAG Reversal step */
(1) Reverse the DAG then we have the new DAG $G' = (V', E')$;
Compute the γ value for each vertex in G;
All source vertices are assigned backward-level 1 and form the set R;
Store all the source vertices into a queue Q;
$V'' = V' - R$;
/* Assigning-backward level step */
(2) **while** V'' is not empty **do**
(2.1) Pop vertex s from Q;
(2.2) **for** each vertex u adjacent from s **do**
(2.2.1) **if** $level(s) + 1 > level(u)$ **then**
u is assigned with the backward-level $level(u) = level(s) + 1$;
end if
(2.2.2) Decrease the *in-degree* of u by one;
(2.2.3) **if** the *in-degree* of u is zero **then**
Insert u into Q and delete it from V'';
end if
end for
end while
/* Positioning with maximum offset step */
(3) **for** each level l (from larger backward-level to backward-level 1) **do**
Vertex with larger γ value is placed earlier in the broadcast
(breaking tie by randomly selecting a vertex);
end for

Fig. 6 Backward maximum offset algorithm

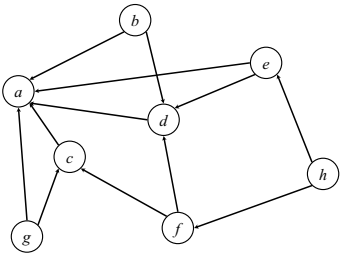


Fig. 7 The reversed DAG of the weighted DAG in Fig. 3

vertex i	a	b	c	d	e	f	g	h
γ(i)	1	0.23	0.2	0.1	0.32	0.15	0.22	0.408
level	4	1	3	3	2	2	1	1
position	1	7	2	3	4	5	8	6

Fig. 8 The backward-level, γ value, and final position of each vertex of the DAG in Fig. 3

out-degree among all the vertices. Overall, both FMOA and BMOA take $O(dn)$ to generate the broadcast schedule where d is the degree of the input DAG.

5.4 The correctness of FMOA and BMOA

In this section, we will show that the proposed heuristics based on the level-oriented strategy produce a broadcast which is a topological ordering of the vertices in the input DAG. To argue this, we first claim the following property about DAG.

Theorem 2. *Given a DAG $G = (V, E)$, suppose $v \in V$ and $v_1, v_2, \ldots, v_l$ are the vertices adjacent from v. Then, there is at least one vertex v_i having no in-edge which is from either v_j or all the vertices reachable from v_j for all $j \neq i$.*

Proof: Assume that for all v_i, there is an in-edge which is from either v_j or all the vertices reachable from v_j where $j \neq i$. W.L.O.G, suppose v_i has a in-edge which is from either v_{i+1} or all the vertices reachable from v_{i+1} for $i = 1, \ldots, l-1$ and v_l has a in-edge which is from either v_k or all the vertices reachable from v_k where $k \in \{1, 2, \ldots, l-1\}$. Then there is a loop in this graph which contradicts to the definition of DAG. □

We use the above theorem to show that the broadcast schedule generated by the level-oriented strategy is a topological ordering of the vertices. We use FMOA to present the argument. Recall the FMOA, the traversed edges will be removed and never traversed again and a vertex becomes a source after all of it's in-edges are removed. Whenever a source v has been traversed, by Theorem 2, at least one new source vertex u will be produced. By keeping the topological ordering among vertices, the level of u is the level of v plus 1. Since all the vertices will be the source only once, such a rule therefore will guarantee the order of two adjacent vertices in the broadcast schedule. Thus, the generated broadcast schedule is a topological ordering of the vertices. We therefore can conclude the following theorem.

Theorem 3. *The broadcast schedule generated by the level-oriented strategy is a topological ordering of the vertices of the input DAG.*

5.5 Bounds on the latency

We now discuss the performance of the proposed heuristics using the level-oriented strategy. Consider an n-vertex weighted DAG $G = (V, E)$. Recall that, for each $v \in V$,

$$\gamma(v) = \sum_{P(s,v) \in \mathbb{P}_v} \prod_{(i,j) \in P(s,v)} p_{ij}.$$

Let $\gamma_{\min}$ denote $\min_{v \in V}\{\gamma(v)\}$. Suppose that f^* is the best broadcast schedule and f is the generated broadcast schedule, respectively. We first give a lower bound (LB) of the latency for the TOBML problem in Lemma 1 and then derive the upper bound (UB) of the latency for the broadcast schedule generated by the level-oriented strategy

in Lemma 2. Last, we compare these bounds with the latency of the best broadcast schedule in asymptotic approach.

Lemma 1. *The latency for the TOBML problem has a lower bound (LB) as*

$$\frac{n(n+1)}{2} \cdot \gamma_{\min}, \tag{3}$$

where $\gamma_{\min} = \min_{v \in V}\{\gamma(v)\}$.

Proof: W.L.O.G., we can rename all the vertices such that $f^*(v_i) = i$ for $i = 1, \ldots, n$. Then the best average latency for accessing all the vertices is

$$\sum_{v_i \in V} f^*(v_i) \cdot \gamma(v_i) = \sum_{v_i \in V} i \cdot \gamma(v_i) \geq \frac{n(n+1)}{2} \cdot \min_{i=1}^{n} \gamma(v_i) = \frac{n(n+1)}{2} \cdot \gamma_{\min} = LB.$$

□

Lemma 2. *If k is the largest level number, then the upper bound of the latency for the broadcast schedule generated by either FMOA or BMOA is* $k \cdot n$.

Proof: Let the vertices assigned to the same level number be a group. Since the largest level number is k, we can form k groups by these k levels. We then number the groups by the level numbers and suppose that each group has m_i vertices for $i = 1, \ldots, k$. Let p_i be the maximum γ value among all the vertices in group i for $i = 1, \ldots, k$. Please note that $p_i \leq 1$ for $i = 1, \ldots, k$ since, for any vertex v in V, $\gamma(v) \leq 1$. Then the average latency for the broadcast schedule generated by either FMOA or BMOA is

$$\begin{aligned}\sum_{v \in V} f(v) \cdot \gamma(v) &\leq p_1 \cdot |m_1| + p_2 \cdot (|m_1| + |m_2|) + \cdots + p_k \\ &\quad \cdot (|m_1| + |m_2| + \cdots + |m_k|) \\ &\leq n \cdot (p_1 + p_2 + \cdots + p_k) \\ &\leq k \cdot n = UB.\end{aligned}$$

□

Having the lower and upper bounds derived in Lemmas 1 and 2, we can derive the asymptotic bound for the ratio, $\rho(n)$, of the latency for the broadcast schedule generated by the level-oriented strategy to the latency of the best broadcast schedule. The bound of the ratio $\rho(n)$ indicates the performance when applying either FMOA or BMOA. From Lemmas 1 and 2, we can derive

$$LB \leq \sum_{v_i \in V} f^*(v_i) \cdot \gamma(v_i) \leq \sum_{v_i \in V} f(v_i) \cdot \gamma(v_i) \leq UB. \tag{4}$$

Thus,

$$\rho(n) = \frac{LB}{UB} = \frac{\frac{n(n+1)}{2} \cdot \gamma_{\min}}{kn} = \frac{n+1}{2k} \cdot \gamma_{\min}. \quad (5)$$

Let ε be $\frac{\gamma_{\min}}{2k}$. We can conclude the following theorem.

Theorem 4. *The latency of the broadcast schedule generated by the level-oriented strategy is within a factor of $\rho(n) = O(n\varepsilon)$ of the latency of a best broadcast schedule, where $\varepsilon = \frac{\gamma_{\min}}{2k}$.*

6 Greedy traversal heuristics

In this section, we provide some algorithms based on the greedy traversal on the DAG to generate the broadcast schedules where the topological ordering among the vertices is kept and the overall latency is minimized. The first one, Revised Topological Sort (RTS), is based on the topological sort [5] to arrange the vertices into the broadcast. The second algorithm, Prior-V, generates the broadcast by considering the γ value of each vertex during the traversal. The third algorithm, Post-V, considers the access probabilities of the posterior vertices of a vertex when traversing the input DAG. Recall that the topological sort traverses the input DAG in a depth-first way. As for algorithms Prior-V and Post-V, both use different schemes respectively to traverse the input DAG and the resulting traversal is not necessarily a depth-first traversal.

6.1 Revised topological sort

We first introduce the RTS algorithm. Recall the topological sort. Suppose vertex v has more than one vertex adjacent from it and these adjacent vertices form a set $ADJ(v)$. Using different order of the vertices in $ADJ(v)$ to continue the traversal makes the results different. To adapt the topological sort to the TOBML problem, whenever a vertex v has been traversed, we make the algorithm always select a vertex u in $ADJ(v)$ to continue the traversal where (1) u has not been traversed and (2) $\gamma(u)$ is the largest among all the vertices in $ADJ(v)$. By [5], the broadcast schedule generated by the RTS is a topological ordering of the vertices. Note that one can also selects the vertex with the smallest γ value to be the next vertex to be traversed intuitively. In our experiments, using either the largest γ value or the smallest γ value leads to a similar latency in average. We use the largest γ value to point out an alternative way to select the vertex to be traversed. We further conjecture that the results of using the largest γ or smallest γ value depend on the structure of the input DAG. This will not be discussed here since it is beyond the scope of this paper.

Consider the DAG in Fig. 3. The corresponding γ value of each vertex is shown in Fig. 5. The RTS starts at vertex a. Vertices b, c, d, e, and g are adjacent from a

and therefore form the set $ADJ(a)$. Since vertex e has the largest γ value among the vertices in $ADJ(a)$, the RTS will traverse vertex e next. After traversing vertex e, vertex h is the only one to be traversed. Then, the RTS will traverse the vertex b because the γ value of b is the second largest one in $ADJ(a)$. Since vertex b has no vertex adjacent to, the RTS will traverse vertex c next. The process continues and the resulting schedule is determined by the finish time of each vertex. The broadcast schedule for the DAG in Fig. 3 is hence *adcfgbeh*.

6.2 *Prior-V* algorithm

In this subsection, we first present algorithm, *Prior-V*, which considers the maximum probability of the prior vertices (i.e. the γ value) for each vertex during the traversal and then show the resulting broadcast is a topological ordering of the input DAG. Recall that we denote the position of a vertex v in the broadcast f as $f(v)$. Algorithm Prior-V starts from the source s. Due to the topological ordering, the source must be assigned to position 1, i.e. $f(s) = 1$.

By deleting s and all the edges incident from it, we can have at least one new source. The next vertex to be visited is the one having the maximum γ value among all the new sources. Suppose vertex v is the next vertex to be visited. Then, we assign vertex v to position 2 (i.e., $f(v) = 2$). The algorithm then again removes v and all the edges incident from v. Again, there is at least one new source. With the set of sources, the same process continues until all the vertices are traversed.

Figure 9 shows a high-level description of algorithm Prior-V. We use a maximum heap H to manage the new sources with the γ value as the key. When traversing a vertex v, v is assigned to the position currently considered and the new sources are inserted into H after vertex v and all the edges incident from v have been deleted. Then, we consider the next position. The next vertex to be visited can be extracted from H which has the maximum γ value in H. The process then repeats until H is empty.

Consider the DAG in Fig. 3. The source a is assigned to the first position. After deleting a from the DAG, we insert new sources c and d into the maximum heap

Input: A weighted DAG, $G = (V, E)$ with source s.
Output: A topological ordering broadcast schedule with minimized latency.

(1)insert the s into H; /* H is a maximum heap with γ value as the key */
(2) $pos = 1$ /* pos: the current position */
(3)**while** $H \neq empty$
  (3.1) extract one vertex with the largest γ value from H;
  (3.2) assign the extracted vertices to position pos;
  (3.3) remove the extracted vertices and all the edges incident from them from the G;
  (3.4) **if** there are new sources after (3.3) **then**
      insert the new sources into H;
  (3.5) $pos = pos + 1$;
  end while

Fig. 9 Prior-V algorithm

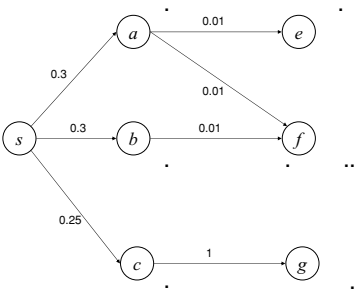


Fig. 10 An example showing that the broadcast generated by Prior-V can be improved by considering the access probabilities of the posterior vertices of a vertex

H. Then algorithm Prior-V extracts c and assigns it to position 2 since $\gamma(c) > \gamma(d)$. After deleting c and all the edges incident from c, the new source g is inserted into H. Again, algorithm Prior-V extracts the source having the largest γ value in H (g in this example) and assigns it to position 3. The process repeats until H is empty and the resulting broadcast schedule is *acgdebfh*.

6.2.1 Correctness

We now show that the broadcast generated by algorithm Prior-V is a topological ordering. Consider the heap H which we use in algorithm Prior-V. Suppose algorithm Prior-V is about to extracts a source from H to assign some position. The prior vertices of all the sources in the current H have been assigned and there is no topological ordering relation among the sources in H. Hence, for the sources extracted from H, all their prior vertices are assigned to earlier positions. We therefore can conclude the following theorem.

Theorem 5. *The broadcast generated by algorithm Prior-V is a topological ordering of the vertices of the input DAG.*

6.3 *Post-V* algorithm

Algorithm Prior-V uses the γ value of a vertex to proceed the traversal on the DAG in order to generate the broadcast schedule. From a global viewpoint, such a greedy assignment does not consider the access probabilities of the vertices on the paths starting from a vertex to the sinks. Thereby, the overall latency of the generated broadcast schedule is obviously not optimized. For instance, consider the DAG in Fig. 10. If we apply algorithm Prior-V, vertices a and b will be assigned to positions before the position assigned to vertex c and vertex g is assigned to the last positions. In fact, vertex g has a higher access probability than e and f and should be assigned to a position prior to the positions of e and f. As a result, it is better that vertex c is assigned before a and b. From such an observation, we propose another algorithm, *Post-V*, which basically considers the access probabilities of the posterior vertices of a vertex. Recall that the posterior vertices of vertex v are the vertices on the paths starting from v.

Fig. 11 The ξ value of each vertex of the DAG in Fig. 3

	a	b	c	d	e	f	g	h
ξ	1	0.23	0.58	0.72	0.9	0.8	0.22	0.408

Algorithm Post-V is similar to algorithm Prior-V in traversing the DAG but uses another parameter ξ instead of the γ value of a vertex to select the source to proceed the traversal. For a vertex v, $\xi(v)$ represents the access probabilities of the posterior vertices of vertex v and can be defined as

$$\xi(v) = \sum_{u \in ADJ(v)} \xi(u) \cdot p_{vu},$$

where $ADJ(v)$ is the set of all the vertices adjacent from v in G and initially each sink vertex v has $\xi(v) = 1$. The ξ value of each vertex can be computed easily. We first reverse the direction of each edge in the DAG as the DAG Reversal step in algorithm BMOA in Section 5.2 and keep the weight on each edge to derive a reverse weighted DAG G'. Then, for each vertex v, we compute the $\xi(v)$ value of v according to the definition given above with a BFS traversal. After computing all the ξ values, we set $\xi(w) = \gamma(w)$ for each sink w. Algorithm Post-V then uses the ξ value of a vertex to arrange the broadcast schedule in the same way as algorithm Prior-V does.

Figure 11 shows the ξ value of each vertex of the DAG in Fig. 3. As mentioned, we set $\xi(w) = \gamma(w)$ for each source w in the reverse DAG since a source in the reverse DAG has no posterior vertex in the original DAG. For example, $\gamma(h) = 0.408$ in Fig. 11. Figure 12 shows the description of algorithm Post-V which first reverses the input DAG and computes the ξ value of each vertex. Then, algorithm Post-V applies algorithm Prior-V but uses the ξ value instead of the γ value when extracting the source vertices from the maximum heap. Using the weighted DAG in Fig. 3, we can derive the resulting broadcast schedule as *adecfhbg*. Since algorithm Post-V uses algorithm Prior-V but with a different parameter to extract the source vertices, the correctness is directly from the correctness of algorithm Prior-V.

6.4 Time complexity

In this section, we discuss the time complexity of algorithm Prior-V and Post-V, respectively. Each operation used in both algorithms, like insertion and deletion on

Input: A weighted DAG, $G = (V, E)$ with source s.
Output: A topological ordering broadcast schedule with minimized latency.

(1) derive the reverse weighted DAG G';
(2) compute the ξ value for each vertex;
(3) apply algorithm Prior-V with ξ value on G;

Fig. 12 Post-V algorithm

the maximum heap H can be done in $O(\log n)$ time where n is the number of vertices in the input DAG. The operation of assigning vertices to channel needs only constant time. Since the heap H contains at most n number of vertices, the time complexity of algorithm Prior-V is $O(n \log n)$. In algorithm Post-V, the computation for the ξ values of all the vertices takes $O(n)$ and it uses algorithm Prior-V with the ξ value instead of the γ value. As a result, the time complexity of algorithm Post-V is also $O(n \log n)$.

6.5 Bounds

Recall that $\gamma_{\min}$ denote $\min_{u \in V}\{\gamma(u)\}$ for a given DAG $G = (V, E)$. Similarly, we let $\gamma_{\max}$ denote $\max_{u \in V}\{\gamma(u)\}$. As shown in Lemma 1, a lower bound on the latency for the TOBML problem is

$$\frac{n(n+1)}{2} \cdot \gamma_{\min}. \tag{6}$$

We now present the upper bound on the latency of the broadcast schedule generated by both algorithm Prior-V and Post-V. We start with the following lemma which will be used to derive the upper bound.

Lemma 3. *Given positive integers $a_1, a_2, \ldots, a_k$. If $a_1 + a_2 + \cdots + a_k = M$, then the following inequality holds,*

$$\prod_{i=1}^{k} a_i \geq M - k, \tag{7}$$

where $1 \leq k \leq M$.

Proof: We will show this by the induction on k. When $k = 1$, obviously $a_1 = M \geq M - 1$. Assume Inequality (7) holds for $k = n$. Then, when $k = n + 1$,

$$\begin{aligned}\prod_{i=1}^{n+1} a_i &= a_{n+1} \prod_{i=1}^{n} a_i \geq a_{n+1} \times (M - n) \\ &\geq 1 \cdot (M - n) \langle \text{since } a_{n+1} \text{ is a positive integer} \rangle \\ &\geq M - (n + 1).\end{aligned}$$

□

To derive an upper bound on the latency, we consider the possibly worst position for a vertex to be assigned. For a vertex v, we let $bl(v)$ be the backward-level of v. The upper bound can be derived as follows.

Theorem 6. *The upper bound on the latency for the broadcast schedule generated by one of the two proposed greedy-traversal heuristics is*

$$\gamma_{\max} \cdot \left\{ n^2 + \frac{3}{2}n - k(k!(n-k))^{\frac{1}{k}} - \frac{k}{2}(n-k)^{\frac{2}{k}} \right\},$$

where n is the number of vertices in the input DAG and k the number of levels.

Proof: Let $f(v)$ be the position of vertex v in the broadcast generated by one of the two algorithms. In order to keep the topological ordering of the vertices in the schedule, the possibly worst position for a vertex v to be assigned is $n - (bl(v) - 1)$. Since there are k levels, $bl(v)$ is between 1 to k for a vertex v. We let b_i denote the set of all the vertices on backward-level $i, i = 1, \ldots, k$. Note that $\{b_1, \ldots, b_k\}$ is a partition of the vertex set V of G. For each backward-level i, the worst position in the broadcast for a vertex in b_i is $n - (i - 1)$.

We first consider the total latency of all the vertices on each backward-level. For each $b_i, i = 1, \ldots, k$, we have

$$\begin{aligned}
\sum_{u \in b_i} f(u) \cdot \gamma(u) &\leq \gamma_{\max} \cdot \sum_{u \in b_i} f(u) \\
&\leq \gamma_{\max} \cdot \{(n - (i-1)) + (n - (i-1) - 1) \\
&\quad + \cdots + (n - (i-1) - (|b_i| - 1))\} \\
&\leq \gamma_{\max} \cdot \left\{ n|b_i| - (i-1)|b_i| - \frac{1}{2}|b_i|^2 + \frac{1}{2}|b_i| \right\}. \qquad (8)
\end{aligned}$$

Then the total latency of all the vertices is:

$$\begin{aligned}
\sum_{i=1}^{k} \sum_{u \in b_i} f(u) \cdot \gamma(u) &\leq \gamma_{\max} \cdot \sum_{i=1}^{k} \left\{ n|b_i| - (i-1)|b_i| - \frac{1}{2}|b_i|^2 + \frac{1}{2}|b_i| \right\} \\
&= \gamma_{\max} \cdot \left\{ n^2 + \frac{3}{2}n - \sum_{i=1}^{k} i|b_i| - \frac{1}{2}\sum_{i=1}^{k} |b_i|^2 \right\}. \qquad (9)
\end{aligned}$$

We now consider each term in Eq. (9) using the Cauchy's Formula and Lemma 3 and can get the following two inequalities:

$$\sum_{i=1}^{k} i|b_i| \geq k \cdot (k!|b_1| \cdot |b_2| \cdots |b_k|)^{\frac{1}{k}} \geq k(k! \cdot (n-k))^{\frac{1}{k}}, \text{ and} \qquad (10)$$

$$\frac{1}{2}\sum_{i=1}^{k} |b_i|^2 \geq \frac{1}{2}k\,(|b_1| \cdot |b_2| \cdots |b_k|)^{\frac{2}{k}} \geq \frac{k}{2}(n-k)^{\frac{2}{k}}. \qquad (11)$$

By applying (10) and (11) to (9), we can conclude the proof. □

7 A refinement algorithm

The heuristics introduced in previous two sections can generate a broadcast schedule which is a topological ordering of the vertices and minimizes the overall latency. We now propose an algorithm which takes a topological ordering broadcast schedule as an input and produces a better broadcast schedule.

Recall that $PRE(v)$ of a vertex v is the set of all the vertices adjacent to v and $ADJ(v)$ is the set of all the vertices adjacent from v. In other words, $PRE(v) = \{u \mid (u, v) \in E\}$ and $ADJ(v) = \{u \mid (v, u) \in E\}$. Let P_v (resp. D_v) be the vertex in $PRE(v)$ (resp. $ADJ(v)$) which position is closest to v. By using the notations, P_v (resp. D_v) $\in PRE(v)$ (resp. $ADJ(v)$) and $f(P_v) = \max_{u \in PRE(v)} f(u)$ (resp. $f(D_v) = \min_{u \in ADJ(v)} f(u)$), where f is the input broadcast schedule.

For each vertex v, we consider the *legal range* of v, $LR(v) = [f(Pv) + 1, f(Dv) - 1]$. Vertex v can be reassigned to any position within its legal range without violating the topological ordering. The basic idea of the proposed algorithm is to consider all the legal positions for each vertex v to look for v's best position in its legal range. When considering a legal position for vertex v, the algorithm will not change the order of the other vertices assigned to a position within $LR(v)$.

Suppose a legal position of v is considered and each vertex u assigned to a position within $LR(v)$ will be assigned to a new position $f'(u)$. We refer such an operation as an *adjustment* of v. Each adjustment changes the total latency of the broadcast schedule. At each legal position of v, we define the *adjustment cost* as

$$AC_{LR}(v) = \sum_{f(u) \in legal(v)} f'(u) \cdot \gamma(u). \tag{12}$$

The algorithm will find the best adjustment of v such that the adjustment cost is the minimum among all the possible adjustments.

Consider the DAG in Fig. 3 and the resulting broadcast schedule "*acdebgfh*" generated by FMOA. The refinement algorithm takes the broadcast schedule generated by FMOA as the input and considers the source a first. The legal range of a, $LR(a)$, is [1,1] and no adjustment will be done. Suppose the algorithm now considers vertex c. Then, $PRE(c) = \{a\}$ and $ADJ(c) = \{f, g\}$. Thus, $P_c = a$ and $D_c = g$ since $6 = f(g) < f(f)$. The legal range of c is $LR(c) = [2, 5]$. The algorithm then considers the adjustment of c at each position between the legal range of c using the adjustment cost defined in (12) and finds the best adjustment of c. In this example, the best adjustment is to insert c at position 5 and the broadcast schedule becomes "*adebcgfh*". Note that the order among the vertices other than c is kept. The next vertex to be considered is vertex d. The legal range of d is $LR(d) = [2, 2]$. This case is the same as the case of vertex a and no adjustment should be done. So, vertex d is still assigned to position 2. The algorithm continues to find the best adjustment for each of the rest vertices. Figure 13 shows a high-level description of the refinement algorithm. Since the refinement algorithm reschedules each vertex within the legal range, the generated broadcast schedule is still a topological ordering of the vertices.

For each vertex, the best adjustment can be found by considering the positions within the legal range one by one from left to right. In order to reduce the amount

Input: A topological-ordering broadcast schedule B and a weighted DAG G.
Output: A refined broadcast schedule having shorter total latency than B.

for $i = 1$ to n **do**
  if v_i is not be explored **then**
    (1) $start = P_{v_i} + 1$;
    (2) $end = D_{v_i} - 1$;
    (3) $R(v_i) = \sum_{j=start}^{end} j \cdot \gamma(v_j)$; $insertIndex = i$;
    (4) **for** $k = i - 1$ down to $start$ **do**
      (4.1) **if** $R(v_i) > AC_{LR}(v_i)$ **then**
        (a) $insertIndex = k$;
        (b) Update $R(v_i) = AC_{LR}(v_i)$;
      end if
    end for
    (5) **for** $k = i + 1$ to end **do**
      (5.1) **if** $R(v_i) > AC_{LR}(v_i)$
        (a) $insertIndex = k$;
        (b) Update $R(v_i) = AC_{LR}(v_i)$;
      end if
    end for
    (6) Inserting v_i to the $insertIndex$ position and remarked v_i is explored;
  end if
end for

Fig. 13 The refinement algorithm

of computation, in our algorithm, we consider the positions left to the vertex and the positions right to the vertex separately. In either part, we can find the best adjustment in the time proportional to the size of that part. Therefore, each vertex can find the best adjustment in $O(n)$ time in worse case. The overall time for the refinement algorithm is thus $O(n^2)$.

8 Simulation

This section presents the simulation results of all the proposed algorithms. We measure the total latency, $\sum_{v \in V} f(v) \cdot \gamma(v)$. All the DAGs in the experiment are generated by the BNGenerator [12] which can generates a DAG randomly. In our experiments, each DAG is of size 1,000 and the degree of each vertex in the DAG ranges from one to seven. The weight on each edge in a DAG is also generated randomly under the constraint that the summation of all the weights from a vertex is less than or equal to one. For each DAG, we run 1,000 different sets of edge weight.

Figure 14 shows the latency when applying FMOA, BMOA, RTS, Prior-V and Post-V algorithms on a DAG with five different sets of edge weights. The Prior-V always generates a schedule with a shorter latency than the others. Recall that Prior-V assigns the vertex having larger γ value to an earlier position in each step and the

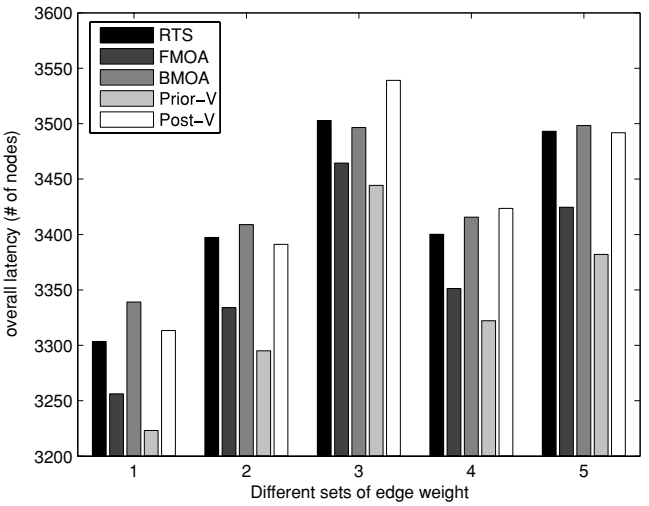


Fig. 14 Latency comparison for the FMOA, BMOA, RTS, Prior-V, and Post-V on a DAG using 5 different sets of edge weights

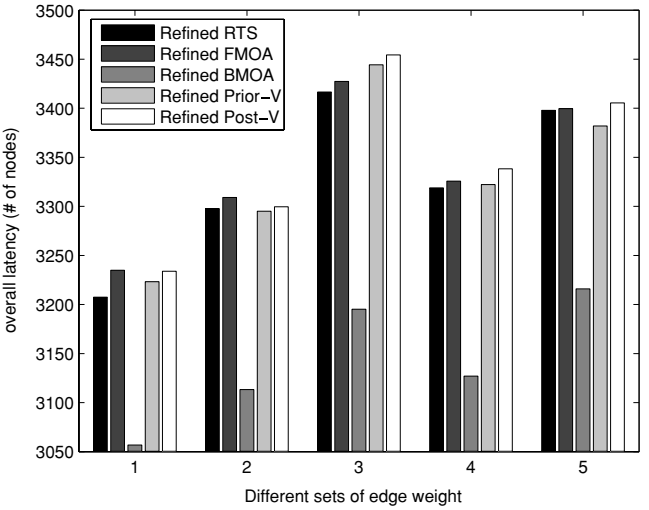
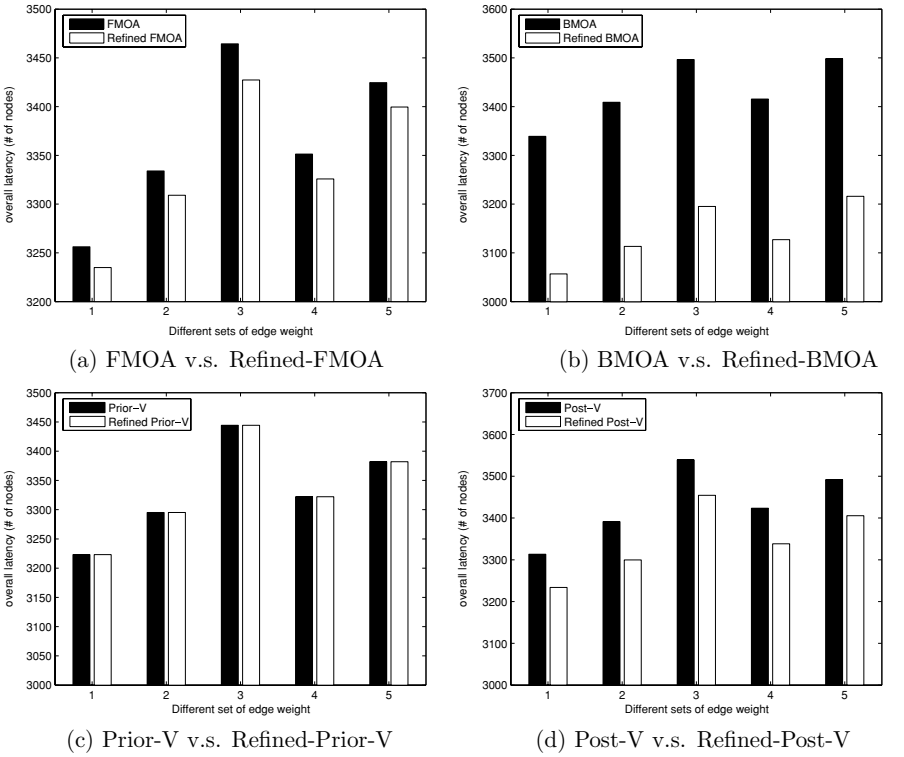


Fig. 15 Latency comparison for applying the refinement algorithm to the broadcast schedules generated by FMOA, BMOA, RTS, Prior-V, and Post-V, respectively on a DAG with 5 different sets of edge weights

γ value of a vertex indicates the probability to refer that vertex. Placing the vertex with larger γ value to an earlier position leads to a shorter overall latency than the algorithms using level-oriented strategy, FMOA and BMOA, due to the limitation on the position posed by the levels. Besides, in most cases, Prior-V performs better



(a) FMOA v.s. Refined-FMOA (b) BMOA v.s. Refined-BMOA

(c) Prior-V v.s. Refined-Prior-V (d) Post-V v.s. Refined-Post-V

Fig. 16 Latency comparison for each algorithm and its corresponding refinement algorithm according to the results in Figs. 14 and 15 (except RTS)

than the other two algorithms using greedy traversal strategy, RTS and Post-V. RTS is basically a depth-first traversal and may neglect some vertices with heavy weight to be traversed in a breadth-first view. As for Post-V, the ξ value of each vertex is considered instead of the γ value. Such a policy does not focus on the γ value of each vertex and hence leads to a longer latency than Prior-V in most cases.

On the other hand, in our experiments, the broadcast schedule generated by BMOA has the longest latency among all the algorithms in most cases. Recall that BMOA uses the backward-level of each vertex and such a level defines the worst position for a vertex. BMOA assigns each vertex to a position far from the worst position. In contrast with BMOA, FMOA considers the best position for each vertex and places each vertex to the position closest to the best position. As a result, FMOB performs better than BMOA.

Figure 15 shows the latency when applying the refinement algorithm to the broadcast schedules generated by FMOA, BMOA, RTS, Prior-V, and Post-V respectively. We consider five different sets of edge weight on a DAG and denote Refined-FMOA, Refined-BMOA, Refined-RTS, Refined-Prior-V, and Refined-Post-V as the refinement algorithms using the broadcast schedule generated by FMOA, BMOA, RTS,

Fig. 17 The average legal ranges when applying the refinement algorithm on the input broadcast schedules generated by FMOA, BMOA, Prior-V, and Post-V, respectively

	FMOA	BMOA	Prior-V	Post-V
1	95	107	83	92
2	95	109	84	95
3	97	108	84	93
4	95	109	83	94
5	95	109	84	92

Prior-V, and Post-V, respectively. Surprisingly, Refined-BMOA generates the broadcast schedule having the shortest latency among these fives algorithms. Figure 16 further shows the latency comparison for each algorithm except RTS. Among the different provided algorithms, Refined-BMOA performs best. Recall that the refinement algorithm adjusts each vertex in the schedule within the legal range of that vertex. Generally speaking, the legal range of each vertex in the schedule generated by BMOA is larger than the one in the schedule generated by the other algorithms. Figure 17 shows this trend. With a larger legal range, the refinement algorithm can produce a better position for each vertex; therefore, leads to a broadcast schedule having a shorter latency. However, it will cost more time to finish the refinement.

9 Conclusions

In this paper, we discuss how to generate the broadcast schedule for a given data set having DAG access pattern in wireless mobile environments. The generated broadcast schedule is a topological ordering of the vertices of the DAG and minimizes the overall latency. We formulate such a problem as the TOBML problem and show that the TOBML problem is NP-complete. Due to the intractability, five different heuristics are presented: FMOA, BMOA, RTS, Prior-V, and Post-V. In addition to, we propose a refinement algorithm to find a better schedule which achieves a shorter total latency as well as keeps the topological ordering among the vertices for the broadcast schedules generated by FMOA, BMOA, Prior-V and Post-V. We further analyze the proposed algorithms on the complexity and latency by giving the lower and upper bounds on the latency of the broadcast schedule. Last, we present the simulation results of these heuristics and compare the performance among them in terms of the latency.

References

1. S. Acharya, M. Franklin, and S. Zdonik, "Balancing push and pull for data broadcasts," in Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, May 1997, pp. 183–194.
2. A. Bar-Noy, J. Naor, and B. Schieber, "Pushing dependent data in clients-providers-servers systems," Wireless Networks, vol. 9, pp. 421–430, 2003.
3. G. Chartrand and O.R. Oellerman, Applied and Algorithmic Graph Theory. McGraw-Hill, 1993.
4. Y. Chehadeh, A. Hurson, and M. Kavehrad, "Object organization on a single broadcast channel in the mobile computing environment," Multimedia Tools and Applications, vol. 9, pp. 69–94, 1999.
5. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms. McGraw-Hill, New York, 2002.
6. S. Even and Y. Shiloah, "Np-completeness of several arrangement problems," Department of Computer Science, Israel Institute of Technology, Haifa, Isreal, Tech. Rep., 1975.

7. S. Hambrusch, C.-M. Liu, W. Aref, and S. Prabhakar, "Query processing in broadcasted spatial index trees," Lecture Notes in Computer Science, vol. 2121, pp. 502–510, 2001.
8. S. Hameed and N. Vaidya, "Efficient algorithms for scheduling data broadcast," ACM/Baltzer Journal of Wireless Networks, vol. 5, no. 3, pp. 183–193, 1999.
9. J.-L. Huang, M.-S. Chen, and W.-C. Peng, "Broadcasting dependent data for ordered queries without replication in a multi-channel mobile environment," in Proceedings of the 19th International Conference on Data Engineering, 2003, pp. 692–694.
10. H.-P. Hung and M.-S. Chen, "On exploring channel allocation in the diverse data broadcasting environment," in ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05), 2005, pp. 729–738.
11. A. Hurson and Y. Jiao, "Data broadcasting in a mobile environment," in Wireless Information Highway. IRM Press, 2004, Ch. 4, pp. 96–154.
12. J.S. Ide, F.G. Cozman, and F.T. Ramos, "Generating random bayesian networks with constraints on induced width," in Proceedings of the 16th Eureopean Conference on Artificial Intelligence, 2004, pp. 323–327.
13. T. Imieliński, S. Viswanathan, and B.R. Badrinath, "Data on air: Organization and access," IEEE Transactions on Knowledge and Data Engineering, vol. 9, no. 3, pp. 353–372, May/June 1997.
14. G. Lee, S.-C. Lo, and A. Chen, "Data allocation on wireless broadcast channels for efficient query processing," IEEE Transactions on Computers, vol. 51, no. 10, 2002.
15. C.-M. Liu, "Broadcasting and blocking large data sets with an index tree," Ph.D. dissertation, Purdue University, West Lafayette, IN, 2002.
16. C.-M. Liu, L.-C. Wang, L. Chen, and C.-J. Chang, "On-demand data disseminating with considering channel interference for efficient shortest-route service on intelligent transportation system," in Proceedings of the 2004 IEEE International Conference on Networking, Sensing and Control, 2004, pp. 701–706.
17. B. Zheng, X. Wu, X. Jin, and D.L. Lee, "Tosa: a near-optimal scheduling algorithm for multi-channel data broadcast," in MEM '05: Proceedings of the 6th international conference on Mobile data management, 2005, pp. 29–37.