

# FlexSched\*: A Parameterized Data Schedule Generator for Multi-Channel Broadcast Systems

Jen-Jou Hung

National Taiwan University of Science and Technology  
Department of Information Management  
No. 43, Section 4, Keelung Road,  
Taipei 10672, Taiwan R.O.C.  
jjhung@juno.cs.ntust.edu.tw

André Seifert

University of Konstanz  
Database Research Group  
P.O. BOX D188  
78457 Konstanz, Germany  
andre.seifert@uni-konstanz.de

## Abstract

*In this paper we study the problem of finding an efficient schedule for broadcasting a given set of data items over multiple physical broadcast channels. The goal of our work is twofold: (a) to minimize the average access latency to fulfill data requests of mobile clients and (b) to provide the ability to tune the period length of the broadcast schedule so as to ease the air-indexing process and to enable clients to determine when the desired data item is broadcast next. To achieve these goals, we propose a parameterized scheduler, called FlexSched\*, which is a polynomial time algorithm and uses a divide-and-conquer approach to efficiently obtain a solution to the problem. To evaluate the robustness and solution quality of FlexSched\*, we conducted an extensive performance study by using several synthetic data sets. The results show that FlexSched\* may significantly outperform existing state-of-the-art data scheduling algorithms with relatively low computational costs.*

## 1 Introduction

Wireless data broadcasting is an efficient and highly scalable way to transmit timely information to a large number of clients, anytime, anywhere, and anyhow. Wireless data broadcast service providers such as Ambient, Microsoft, or SkyTel try to benefit from the prevalence of portable mobile devices and the widespread use of new communication technologies and continuously disseminate interesting private and public information to mobile clients. Since available mobile client and network bandwidth resources are scarce, broadcast service providers need to be concerned with two critical issues when deciding on the content and structure of the broadcast program, namely (a) *access efficiency* and (b) *power expenditure*.

The access efficiency of a broadcast system is determined by the broadcast schedule which specifies when each

data item is to be broadcast. Its power expenditure depends on the provision of indexing information about the arrival times of the scheduled data items. Since air-indexes need to be broadcast prior to data items, an *off-line*, rather than an on-line scheduling algorithm is required. In addition, it is desirable that the algorithm produces *periodic* schedules. Periodicity of a broadcast schedule is a very useful property for at least two reasons: First, it allows the server to index an entire schedule period once and then to (re-)use the resulting air-indexes in multiple broadcast cycles without any modifications. Second, it allows clients to determine the broadcast time of the requested data items by inspecting locally cached replicas of the air-indexes which helps them to save valuable tuning time.

Based on these findings, we propose a novel parameterized off-line scheduling algorithm, called FlexSched\*, which can be adjusted to optimize the access latency with the schedule period length bounded by a given limit  $L$ . The intuition behind proposing an adjustable scheduling algorithm is the observation that neither the *access latency* nor *schedule period length* can be optimized without adversely impacting the other. The adversarial relationship is straightforward to explain: If the schedule period length is set to the lowest possible value, i.e., each data item is allowed to appear at most once during each broadcast cycle, a *flat* broadcast schedule can only be generated which produces suboptimal access latencies when the distribution of the demand probabilities of the data items is skewed. If, on the other hand, the period length of the schedule is allowed to take on values larger than the lowest possible one, data items may appear more than once during a schedule period, i.e., a *skewed* broadcast schedule can now be produced. Clearly, the larger the allowed schedule period length, the higher the scheduling freedom to minimize the access latency of the schedule. The drawback of larger schedule periods, however, is that more space is required in order to store the

entire schedule and its associated air-indexes. To be able to accommodate the different requirements of the different applications, we propose FlexSched\*, which is able to facilitate schedule period bounded performance tuning.

The remainder of the paper is organized as follows: In Section 2 related work will be reviewed, followed by the definition of some notations and terminologies in Section 3. Section 4 describes the FlexSched\* scheduling algorithm in detail and experimental results follow in Section 5. A summary of the paper is presented in Section 6.

## 2 Related Work

The problem of finding latency-minimal broadcast schedules has been extensively discussed by both the network and database communities, but from slightly different perspectives. Most proposals from the network community as well as ours adopt the square-root rule [1, 9] for constructing latency-minimal broadcast schedules. However, it may not be possible to satisfy all the conditions imposed by the square-root rule at the same time [8]. To overcome this difficulty, the researchers from the network community [1, 2, 9] take the approach of scheduling data items according to their exact optimal broadcast bandwidth allocation. A drawback of this approach is that the resulting schedule is usually not periodic, making air-indexing very inefficient. The database community as well as ourselves explores the other alternative, namely constructing latency-minimal periodic broadcast schedules, while giving up on the satisfaction of the exact average optimal broadcast frequency allocation. Previous approaches [3, 4, 5, 6] proposed to generate *flat schedules* for multi-channel broadcast systems. An advantage of flat schedules is obviously their short period lengths. However, if the demand probabilities of the data items are different, flat scheduling will obviously produce only suboptimal periodic schedules. FlexSched\* attempts to rectify the problems of existing methods by proposing a parameterized scheduling algorithm that attempts to produce latency-minimal broadcast schedules whose period lengths are bounded by a specified limit, thus allowing system engineers to trade-off between access latency and schedule period length.

## 3 Preliminaries

### 3.1 Notations and Definitions

Let a set  $D = \{d_1, d_2, \dots, d_n\}$  of  $N$  data items represent a database to be broadcast over a set  $C = \{c_1, c_2, \dots, c_m\}$  of  $M$  physical broadcast channels. Each data item  $d_i$  has a demand probability  $p_i$ , and a length  $l_i$ . Clearly,  $\sum_{i=1}^N p_i = 1$ . Each physical broadcast channel  $c_j \in C$  has a bandwidth

capacity of  $bc_j$  and the total bandwidth of all channels is given by  $BC = \sum_{j=1}^M bc_j$ . FlexSched\* first partitions the data items in  $D$  into an ordered set  $G = \{g_1, g_2, \dots, g_m\}$  of  $M$  data groups with each data group  $g_j \in G$  being associated to some broadcast channel  $c_j$ .

In order to minimize the access latency, FlexSched\* attempts to build a *skewed broadcast schedule* in which the data items are broadcast according to their demand probabilities and lengths. To achieve this, FlexSched\* partitions the data items in each data group  $g_j$  into an ordered set  $G_j = \{g_{j,1}, g_{j,2}, \dots, g_{j,k}\}$  and assigns to each data group  $g_{j,k}$  a broadcast frequency  $bf_{j,k}$ , which represents the number of data items of  $g_{j,k}$  that will be broadcast during a *minor broadcast cycle* (MIBC). An MIBC is a finite sequence  $S_{mibc} = \langle d_1, d_2, \dots, d_o \rangle$  of  $O$  instances of data items such that for each data group  $g_{j,k} \in G_j$  there are  $bf_{j,k}$  instances of distinct data items in  $S_{mibc}$ .

A broadcast schedule of broadcast channel  $c_j$  is called *periodic* if there exists for each data group  $g_{j,k} \in G_j$  and each data item  $d_i \in g_{j,k}$  a number  $p$  such that  $d_i + p = d_i$ . That is, if an instance of data item  $d_i$  is scheduled at time  $t_i$ , then  $d_i$  is also scheduled at time  $t_i + p$ . The period length of a schedule is the smallest number  $p$  for which the above property holds and is given by:

$$p = lcm(pl_{j,1}, pl_{j,2}, \dots, pl_{j,k}), \quad (1)$$

where  $pl_{j,k}$  denotes the period between successive instances of the same data item of data group  $g_{j,k}$  amounts to  $pl_{j,k} = |g_{j,k}|/bf_{j,k}$  MIBCs, and  $|g_{j,k}|$  denotes the cardinality of  $g_{j,k}$ . The function  $lcm()$  returns the least common multiple of period lengths of all data groups in  $G_j$ .

### 3.2 Theoretical Foundation

To achieve optimal access latencies in a multi-channel broadcast environment, the following two conditions need to be satisfied:

- According to square-root rule [1, 9], the optimal fraction  $f_i$  of broadcast channel bandwidth allocated to a data item  $d_i$  is given by:

$$f_i = \frac{l_i \cdot \sqrt{p_i/l_i}}{\sum_{i=1}^N \sqrt{p_i \cdot l_i}}. \quad (2)$$

- Two consecutive instances of the same data item  $d_i$  should be spaced equally apart [1, 9] with spacing  $s_i$  being equal to  $l_i/f_i$ .

While the optimal allocation of the broadcast channel bandwidth to the data items does not pose any problems, the satisfaction of the equal-spacing requirement, however, may not always be realizable [8]. Nevertheless, the above scheduling conditions present a valuable theoretical basis for evolving FlexSched\*.

## 4 The FlexSched\* Algorithm

FlexSched\* initially puts the set  $D = \{d_1, d_2, \dots, d_n\}$  of  $N$  data item in a descending order of their probability-length ratios. The resulting sorted sequence  $S = \langle d_1, d_2, \dots, d_n \rangle$  is then partitioned into an ordered set  $G = \{g_1, g_2, \dots, g_m\}$  of  $M$  groups by applying the optimal bandwidth allocation formula (see Equation 2) with each group  $g_j$  being associated with a physical broadcast channel  $c_j$  with bandwidth capacity  $bc_j$ . The result is a segmentation

$$\underbrace{d_1, \dots, d_{b_1}}_{g_1}, \underbrace{d_{b_1+1}, \dots, d_{b_2}}_{g_2}, \dots, \underbrace{d_{b_{m-1}+1}, \dots, d_n}_{g_m}$$

with  $p_{b_i}/l_{b_i} \geq p_{b_j}/l_{b_j}$  whenever  $b_i < b_j$ .

According to the square-root rule, the minimum overall data access latency is achieved when the spacing for each data item  $d_i$  is proportional to  $\sqrt{\frac{l_i}{p_i}}$ . Thus, to minimize the overall access latency, the relative optimal broadcast frequency of each data item  $d_i \in g_j$ , denoted by  $b_{f_i}^{opt}$ , is set to  $\sqrt{p_i/l_i}/\sqrt{p_{b_{j-1}+1}/l_{b_{j-1}+1}}$ , where  $b_{j-1} + 1$  is the identifier of the data item that has the highest probability-length ratio among the data items of group  $g_j$ . This observation is exploited by FlexSched\* in the next step of the algorithm. Here, it partitions the data items in each group  $g_j$  into an ordered set  $G_j = \{g_{j,1}, g_{j,2}, \dots, g_{j,k}\}$  of  $K$  singleton groups. That is, each data item  $d_i \in g_j$  is assigned to its own data group  $g_{j,k}$ ,  $1 \leq k \leq |g_j|$ . Besides, FlexSched\* keeps track of  $g_{j,k}$ 's optimal relative broadcast frequency  $b_{f_{j,k}}^{opt}$ , demand probability  $p_{j,k}$ , and length  $l_{j,k}$ , which are equal to  $d_i$ 's optimal relative broadcast frequency  $b_{f_i}^{opt}$ , demand probability  $p_i$ , and length  $l_i$ , respectively.

FlexSched\* assigns to each singleton group  $g_{j,k} \in G_j$  an integer-valued broadcast frequency  $b_{f_{j,k}}$  of 1 since it allows for instant generation of a periodic broadcast schedule with minimal period length. Such an initial "seed" schedule obviously will produce optimal access latencies only if both the lengths of the data items and their demand probabilities follow a uniform distribution. For *skewed distributions* in both values, however, the initial seed schedule will perform poorly. The reason why the initial schedule typically produces suboptimal performance results is related to the divergence between the groups' optimal broadcast frequencies and their integer-valued ones. This observation motivates us to use the concept of *group merging* to improve upon the overall scheduling performance since it helps us both to narrow prevalent frequency gaps and to control the increase of the period length of the schedule.

In the group merging algorithm as shown in Algorithm 1, FlexSched\* first initializes the variables  $l_j$  and  $q_j$ , where  $l_j$  represents the MIBC length of the initial broadcast schedule and  $q_j$  is the sum of the weighted inter-arrival times of the

**Algorithm 1:** Group Merging Procedure

---

```

1  $q_j \leftarrow \sum_{k=1}^{|G_j|} \frac{|g_{j,k}| \cdot p_{j,k}}{b_{f_{j,k}}}$ 
2  $l_j \leftarrow \sum_{k=1}^{|G_j|} \frac{b_{f_{j,k}} \cdot l_{j,k}}{|g_{j,k}|}$ 
3 repeat
4   for  $k \leftarrow 4; k \leq |G_j|; k \leftarrow k + 2$  do
5      $b_{f_{k-2,k-1}} \leftarrow \text{round}(b_{f_{k-2}}^{opt} + b_{f_{k-1}}^{opt});$ 
6      $b_{f_{k-1,k}} \leftarrow \text{round}(b_{f_{k-1}}^{opt} + b_{f_k}^{opt});$ 
7     calculate  $\Delta_{k-2,k-1}$  and  $\Delta_{k-1,k}$ ;
8     if  $\text{evalMerge}(\Delta_{k-2,k-1}, \Delta_{k-1,k},$ 
9        $|g_{j,k-2}| + |g_{j,k-1}|, b_{f_{k-2,k-1}})$  then
10        $p_{j,k-2} \leftarrow p_{j,k-2} + p_{j,k-1};$ 
11        $l_{j,k-2} \leftarrow l_{j,k-2} + l_{j,k-1};$ 
12        $b_{f_{k-2}}^{opt} \leftarrow b_{f_{k-2}}^{opt} + b_{f_{k-1}}^{opt};$ 
13       update  $q_j$  and  $l_j$  if necessary
14     if  $\text{evalMerge}(\Delta_{k-1,k}, \Delta_{k-2,k-1},$ 
15        $|g_{j,k-1}| + |g_{j,k}|, b_{f_{k-1,k}})$  then
16        $p_{j,k-1} \leftarrow p_{j,k-1} + p_{j,k};$ 
17        $l_{j,k-1} \leftarrow l_{j,k-1} + l_{j,k};$ 
18        $b_{f_{k-1}}^{opt} \leftarrow b_{f_{k-1}}^{opt} + b_{f_k}^{opt};$ 
19       update  $q_j$  and  $l_j$  if necessary;
20        $k \leftarrow k + 1$ 
21 until (no pair of data groups has been merged)

```

---

data items in terms of the unit of MIBCs. Note that the product of  $l_j$  and  $q_j$ , if divided by 2, corresponds to the average access latency of the initial broadcast schedule.

Hereafter the merge procedure follows. It is implemented as a conditional loop that is active as long as at least one pair of adjacent data groups is merged during an entire merge loop iteration. To decide on merging data groups, FlexSched\* considers triples  $(g_{j,k-2}, g_{j,k-1}, g_{j,k})$  of successive data groups. Clearly, each such triple contains two pairs of adjacent data groups  $(g_{j,k-2}, g_{j,k-1})$  and  $(g_{j,k-1}, g_{j,k})$ , and therefore each such pair is regarded as a merging candidate by FlexSched\*. More specifically, for each pair of adjacent data groups  $(g_{j,k-2}, g_{j,k-1})$  and  $(g_{j,k-1}, g_{j,k})$ , FlexSched\* first computes their "optimal" integer-valued broadcast frequencies  $b_{f_{k-2,k-1}}$  and  $b_{f_{k-1,k}}$ . Then, it uses Formula 3 to calculate the differences  $\Delta_{k-2,k-1}$  and  $\Delta_{k-1,k}$  between the approx. overall average access time of the schedule before and after merging groups  $(g_{j,k-2}, g_{j,k-1})$  and  $(g_{j,k-1}, g_{j,k})$ , respectively. Based on the accumulated data, FlexSched\* subsequently decides by using the function  $\text{evalMerge}()$  as shown in Algorithm 2 whether to merge data groups  $(g_{j,k-2}, g_{j,k-1})$  and  $(g_{j,k-1}, g_{j,k})$ , respectively, or better keep them all split apart.

After no more data groups are eligible for merging, FlexSched\* proceeds by constructing the broadcast sched-

$$\Delta_{k-2,k-1} = \frac{1}{2} \cdot \left( \frac{(|g_{j,k-2}| + |g_{j,k-1}|) \cdot (p_{j,k-2} + p_{j,k-1})}{\text{round}(bf_{k-2}^{opt} + bf_{k-1}^{opt})} - \frac{|g_{j,k-2}| \cdot p_{j,k-2}}{bf_{j,k-2}} - \frac{|g_{j,k-1}| \cdot p_{j,k-1}}{bf_{j,k-1}} \right) - \left( \frac{(l_{j,k-2} + l_{j,k-1}) \cdot bf_{k-2,k-1}}{|g_{j,k-2}| + |g_{j,k-1}|} - \frac{l_{j,k-2} \cdot bf_{j,k-2}}{|g_{j,k-2}|} - \frac{l_{j,k-1} \cdot bf_{j,k-1}}{|g_{j,k-1}|} \right) \quad (3)$$

---

**Algorithm 2:** Auxiliary function

---

```

1 function evalMerge( $\Delta_{i,j}, \Delta_{j,k}, |g_{j,k}|, bf_{j,k}$ )
2 begin
3   if  $\Delta_{i,j} > 0$  and  $\Delta_{i,j} > \Delta_{j,k}$  and
   ( $|g_{j,k}| \bmod bf_{j,k} = 0$  and schedule period
   length does not exceed limit L) then
4     return true
5   else
6     return false
7 end

```

---

ule for each physical broadcast channel  $c_j$ . To this end, FlexSched\* applies Algorithm 3 which logically divides the broadcast schedule into  $p$  MIBCs and transmits data items from the data groups belonging to  $R_j$  in a round-robin fashion, where  $R_j$  denotes the set of groups that are left after the group merging procedure is completed. Due to the iterative nature of the algorithm, it produces an infinite periodic schedule unless either the broadcast content, the demand probabilities of the data items, or their lengths change.

---

**Algorithm 3:** Schedule Construction Procedure

---

```

1  $p \leftarrow \text{lcm}(pl_{j,1}, pl_{j,2}, \dots, pl_{j,k}), \forall pl_{j,k} : g_{j,k} \in R_j;$ 
2 repeat
3   for  $i \leftarrow 0$  to  $p - 1$  do
4     foreach data group  $g_{j,k}$  in  $R_j$  do
5       for  $j \leftarrow 1; j \leq bf_{j,k}; j \leftarrow j + 1$  do
6          $k \leftarrow (i \cdot bf_{j,k}) \bmod |g_{j,k}|;$ 
7         if  $k + j > |g_{j,k}|$  then
8            $k \leftarrow (k + j) - |g_{j,k}|$ 
9         else
10           $k \leftarrow k + j$ 
11          broadcast  $k$ -th data item of group  $g_{j,k}$ 
12 until (broadcast content or demand probability or
   length of any data item has changed)

```

---

## 5 Performance Evaluation

We now present performance results for FlexSched\* and other state-of-the-art scheduling algorithms obtained

on synthetic data sets. The synthetic data sets have been generated by using a *random length distribution* with the item lengths randomly distributed from 10 to 100 with uniform probability and by assuming that the demand probability follows a *Zipf distribution* with the parameter  $\theta$  set to 0.8. We produced synthetic workloads emulating various database sizes in the range of 500 to 100,000. If not otherwise stated, the experiments were run with a default database size of 10,000 and the number of physical broadcast channels was set to 4.

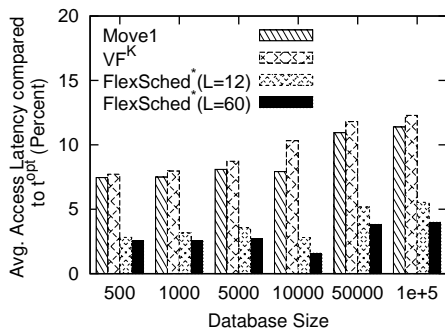
To study the performance of FlexSched\* in comparison to other state-of-the-art scheduling algorithms, we implemented the Move1 [3] and Variant Fanout with Constraint  $K$  (VF<sup>K</sup>) [6] scheduling algorithms. We opted for Move1 for comparison since Move1 has been empirically shown to be the best-to-date heuristic for scheduling data item with non-uniform lengths on an arbitrary number of broadcast channels. VF<sup>K</sup> has been chosen since it has been left out of the performance study in [3] and has been shown to achieve good performance results for the uniform length case [7]. To be able to assess the quality of the solutions obtained by these three scheduling algorithms, we compare them to the optimal overall average access time, denoted  $t^{opt}$ , which is obtained by the following formula [9]:

$$t^{opt} = \frac{1}{2} \cdot \left( \sum_{i=1}^N \sqrt{p_i \cdot l_i} \right)^2 \quad (4)$$

### 5.1 Experiment 1. Effect of the Database Size

In the first experiment, we studied the effect of the database size on the average access latency of the broadcast schedule. Figure 1 shows the relative average access latency of VF<sup>K</sup>, Move1, FlexSched\*(L=12) and FlexSched\*(L=60) compared to  $t^{opt}$ , where FlexSched\*(L=12) and FlexSched\*(L=60) stand for FlexSched\* with the limit  $L$  on the period length set to 12 and 60, respectively. Note that we ran our experiments using various values for  $L$  in the range between 1 and  $\infty$ . Of these values, we selected  $L = 12$  and 60 for result presentation since FlexSched\*'s performance is then close to its worst and best case performance, respectively. By observing the results, we notice that the performance of

FlexSched\* is very close to that of  $t^{opt}$ . The average performance gap of FlexSched\*(L=12) and FlexSched\*(L=60) compared to  $t^{opt}$  is, only 4% and 3%, respectively, while the average performance penalty of Move1 and VF<sup>K</sup> compared to  $t^{opt}$  is about 9% and 10%, respectively. The reason for FlexSched\*'s outperformance is related to the fact that it generates skewed broadcast schedules, whereas VF<sup>K</sup> and Move1 rely on flat scheduling. Remember that in flat scheduling all data items are of the same broadcast frequency. Such a scheduling behavior obviously wastes the channel bandwidth when the distribution of the demand probabilities is skewed.



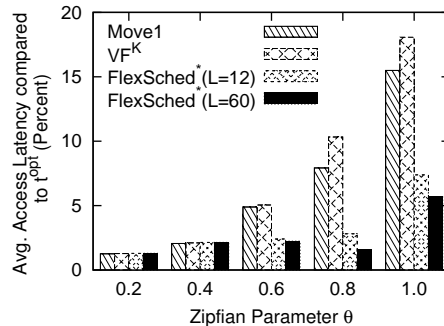
**Figure 1. Relative performance degradation compared to  $t^{opt}$  for various database sizes.**

## 5.2 Experiment 2. Effect of the Degree of Data Access Skewness

Next, we investigated the effect of skewness in the distribution of the demand probabilities of the data items on the average access latency of the scheduling algorithms. To do so, we varied the value of the Zipfian parameter  $\theta$  in the range of 0.2 to 1.0. The results are shown in Figure 2. From Figure 2, we observe that when  $\theta$  is small, the performance of all algorithms is close to that of  $t^{opt}$ . As  $\theta$  increases, i.e.,  $\theta > 0.4$ , the performance difference between FlexSched\* and other two algorithms starts to increase. For example, when  $\theta$  is 0.4, the average access latency of FlexSched\*(L=60) is about 1% lower than that of Move1 and VF<sup>K</sup>; as  $\theta$  increases to 1.0, the performance gap widens to about 10% and 13%, respectively. The reason is again related to the fact that Move1 and VF<sup>K</sup> do merely generate flat schedules.

## 6 Summary

In this paper, we studied the problem of constructing latency-minimal periodic broadcast schedules for multi-channel broadcast systems when the data items are not necessarily equal-sized and their access probabilities are a pri-



**Figure 2. Relative performance degradation compared to  $t^{opt}$  for various values of  $\theta$ .**

ori known by the broadcast server. We presented a novel parameterized scheduling algorithm, called FlexSched\*, which can be easily adjusted to optimize the access latency with the period length of the schedule bounded by a given limit. Through experiments, we have demonstrated that FlexSched\* outperforms state-of-the-art scheduling algorithms under a wide range of system settings and workload conditions. In addition, the results show that FlexSched\* yields performance very close to the optimum and it is very effective in adjusting the trade-off between access latency and schedule period length.

## References

- [1] M. H. Ammar and J. Wong. The Design of Teletext Broadcast Cycles. *Performance Evaluation*, 5(4):235–242, 1985.
- [2] M. H. Ammar and J. Wong. On the Optimality of Cyclic Transmission in Teletext Systems. *IEEE Trans. on Communications*, 35(1):68–73, 1987.
- [3] E. Ardizzoni, A. A. Bertossi, M. C. Pinotti, S. Ramaprasad, R. Rizzi, and M. V. S. Shashanka. Optimal Skewed Data Allocation on Multiple Channels with Flat Broadcast per Channel. *IEEE Trans. on Computers*, 54(5):558–572, 2005.
- [4] C.-H. Hsu, G. Lee, and A. L. P. Chen. A Near Optimal Algorithm for Generating Broadcast Programs on Multiple Channels. In *CIKM 2001*, pages 303–309, 2001.
- [5] D. Katsaros and Y. Manolopoulos. Broadcast Program Generation for Webcasting. *Data Knowl. Eng.*, 49(1):1–21, 2004.
- [6] W.-C. Peng and M.-S. Chen. Efficient Channel Allocation Tree Generation for Data Broadcasting in a Mobile Computing Environment. *Wireless Networks*, 9(2):117–129, 2003.
- [7] A. Seifert and J.-J. Hung. FlexSched: A Flexible Data Schedule Generator for Multi-Channel Broadcast Systems. Technical Report 211, University of Konstanz, 2005.
- [8] N. Vaidya and S. Hameed. Improved Algorithms for Scheduling Data Broadcast. Technical report, Texas A & M University, College Station, TX, USA, 1996.
- [9] N. H. Vaidya and S. Hameed. Scheduling Data Broadcast in Asymmetric Communication Environments. *ACM Wireless Networks*, 5(3):171–182, 1999.