Broadcast Scheduling with Data Bundles

Fangfei Chen[a], Diego Pizzocaro[b], Matthew P. Johnson[a], Amotz Bar-Noy[c], Alun Preece[b] and Thomas La Porta[a]

[a]Dept. of Computer Science and Engineering, The Penn State University, US
[b]School of Computer Science and Informatics, Cardiff University, UK
[c]Dept. of Computer Science, Graduate Center, City University of New York, US

ABSTRACT

Broadcast scheduling has been extensively studied in wireless environments, where a base station broadcasts data to multiple users. Due to the sole wireless channel's limited bandwidth, only a subset of the needs may be satisfiable, and so maximizing total (weighted) throughput is a popular objective. In many realistic applications, however, data are dependent or correlated in the sense that the joint utility of a set of items is not simply the sum of their individual utilities. On the one hand, *substitute* data may provide overlapping information, so one piece of data item may have lower value if a second data item has already been delivered; on the other hand, *complementary* data are more valuable than the sum of their parts, if, for example, one data item is only useful in the presence of a second data item.

In this paper, we define a *data bundle* to be a set of data items with possibly nonadditive joint utility, and we study a resulting broadcast scheduling optimization problem whose objective is to maximize the utility provided by the data delivered.

1. INTRODUCTION

Broadcast scheduling is a well-studied wireless information delivery paradigm in which one server broadcasts messages to all users over a broadcast channel. Users may arrive or depart the problem setting at different times and with various data needs, and therefore schedules are made in order to favor as many users as possible, to the greatest degree possible. Recent example applications include scheduling data downloads in mobile or vehicular networks. In one scenario, a single base station serves multiple mobile clients who want to download information from the base station as they are traveling.

This problem is usually cast as a single machine scheduling problem in which the download of an information item and the broadcast channel play the roles of *job* and *machine*, respectively. We assume that the download time or *job size* for each piece of information is a property of the information item and does not depend on when the download is scheduled. However, since users are traveling, this download has to be scheduled within a certain time window. The start and end of this time window are referred to as release time and deadline. Different pieces of information may have different levels of importance or *utility* values. A common objective is then to maximize the total utility.

The simplest way of modeling the utility of multiple items is as the sum of the individual utility values, but in many applications this is not appropriate. Adopting terminology from economics, we will discuss substitute data and complementary data, by analogy to substitute goods and complementary goods. First, *substitute* data may provide overlapping information or the same information from different perspectives. For example, turn-by-turn instructions and a map can both lead one to a destination, so the value of the two together may reasonably be modeled as less than the sum of their individual utility values; that is, the value e.g. of turn-by-turn instructions is lower if one already has the map. Conversely, *complementary* data increase in value when joined together. For example, the four quadrants of a map may be more useful than four times the value of one quadrant. In economics, complementary products are defined as "those that tend to be consumed jointly in order to satisfy a particular need, while substitute products are those that satisfy the same need and tend to be consumed separately".[1] Classic examples of substitutes and compliments are (respectively) coffee and tea on the one hand, and coffee and sugar on the other.[1]

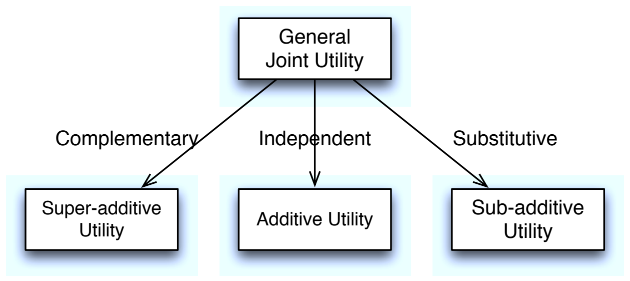


Figure 1: Joint utilities.

We define a *data bundle* to be a set of data items with a well defined joint utility function, not necessarily additive. If not, then data items in a bundle are either substitute or complementary. In terms of utility functions, their joint utility is then either *subadditive* or *superadditive*. Figure 1 illustrates the possibilities.

Our algorithms make decisions based on an item's *marginal* utility, which is the total utility increase that would occur if the item were added to an existing (partial) schedule (and any conflicting items were removed). We first adapt an existing algorithm from the additive utility setting. The algorithm schedules job *instances* (all possible intervals in which the job can be scheduled) in the order of increasing interval *end times* and replaces scheduled instances according to a certain criterion. Then we propose another, simpler algorithm that chooses among jobs rather than job instances, taking them in the order of decreasing marginal utility over processing time. We also discuss a special case of this problem that can be solved optimally using dynamic programing.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 formally defines the problem and proves its hardness. Section 4 presents two heuristic algorithms which are then evaluated in Section 5. Finally, Section 6 concludes the paper.

2. RELATED WORK

This work concerns mobile data access[2] and specifically, vehicle-roadside data access. Systems such as MobiEyes[3] and Thedu[4] have been developed in order to either collect data from or deliver data to moving vehicles. As in our problem, vehicles can only download data within a time window that is determined jointly by transmission range, distance to the RSU, and the vehicle's speed, and multiple vehicles compete for the exclusive use of the common broadcast channel. The task is to schedule access to data by multiple vehicle, subject to the time constraints. Jiang et al.[5] studied a variant in which data items are pushed to vehicles periodically; Xu et al.[6] investigated another variant in which data are only pushed on-demand, as in our problem. The novel aspect of our problem in this paper is that the utility value of a collection of data items need not be the sum of their individual values.

On the theoretical side, the problem with additive utility is widely studied. It can be interpreted as a single machine scheduling problem, which is denoted by $1|r_i|\sum w_i U_i$ in the standard notation and is NP-hard. Bar-Noy et al.[7] provided an LP-rounding 3-approximation algorithm and a combinatorial $3+2\sqrt{2}$-approximation algorithm. Later, Berman and Dasgupta[8] and Bar-Noy et al.[9] gave combinatorial $2/(1-\epsilon)$-approximation algorithms. These two algorithms do not appear to apply readily to the nonadditive setting, however.

Nonadditive utilities have also received attention in the field of *sensor networks*. Sensors might indeed make a nonlinear contribution to the completion of a particular sensing task. For example, Rowaihy et al.[10] considers two different sensing tasks, a detection and a localization task, using a nonlinear utility function in each case to evaluate the quality of bundle of sensors when assigned to a given task. In particular for a localization task, two audio sensors are needed to record an audio event. The quality or accuracy of the result depends on the relative positions of the two sensors to the location event. Let d_1, d_2 be the distances from the sensors to the event, and let θ be the angle formed by the two corresponding rays. Then the uncertainty of the measurement[11] is: $\frac{d_1 d_2}{\sin\theta}$. Therefore the quality will be maximized when $\theta = \pi/2$ and the distances are as small as possible.

Nonadditive utility for a set of items is common in the study of *combinatorial auctions*, in which bidders can express preferences on bundles or *combinations* of items.[12] Given a fixed supply of goods, the goal of the winner determination problem[13] is to maximize revenue earned from the sale of disjoint item combinations. This is a difficult problem, and not just because the number of bids can be exponential in the number of items. There has been much research devoted to this problem, in the AI, algorithm-engineering, and other communities.[14] A view sometimes expressed in the literature is that combinatorial auctions can in practice provide good approximate solutions within reasonable time for problem instances of reasonable size (in terms of number of bids).[14] In our case the number of bids (i.e. possible schedule for each data item) could be very large and so we do not adopt the model of combinatorial auctions.

3. PROBLEM FORMULATION

In this section, we formaly define the problem and prove it to be NP-hard.

3.1 Problem Formulation

Let $D = \{D_i : i = 1 \dots n\}$ denote the set of data items. Each D_i is associated with a weight w_i and a download duration or *processing time* p_i. In order to succeed, D_i must be scheduled after a release time r_i and must finish before a deadline d_i. Disjoint subsets $B_1, B_2, ...B_m \subseteq D$ ($B_j \cap B_k = \emptyset, \forall j, k$) are called *data bundles*. Each data bundle B_k is associated with a joint utility function $U_k(x)$. For a single data item $D_i \in B_k$, $U_k(D_i) = w_i$.

Let x_{it} denote whether D_i is scheduled to start at time t. The objective function is to maximize the total joint utility among all bundles. The problem can be formulated as in Table 1:

$$\max \sum_k U_k(\bigcup_{D_i \in B_k} D_i \sum_{t=r_i}^{d_i - p_i} x_{it}) \tag{1}$$

subject to

$$\sum_t x_{it} \leq 1, \qquad \forall\, i \tag{2a}$$

$$\sum_i \sum_{u=t-p_i+1}^{t} x_{iu} \leq 1, \qquad \forall\, t \tag{2b}$$

where

$$x_{it} = \begin{cases} 1 & \text{if } D_i \text{ is scheduled at time slot } t, \\ 0 & \text{otherwise} \end{cases}$$

Table 1: Problem formulation.

The first set of constraints ensure that for each data d_i, we schedule at most one instance of it. The second set of constraints ensure that the schedule has no conflicts.

3.2 NP-hardness Result

We now present a hardness result for the general problem.

THEOREM 3.1. *Problem 1 is NP-hard, even in the special case of additive utilities.*

Proof. Given is an instance of the 0/1 knapsack problem with knapsack bound W and n items, each having a weight w_i and a size p_i. An instance of Problem 1 is then constructed by setting $r_i = 0$, $d_i = W$ for all i and $U_k(x) = \sum_{D_i \in x} w_i$ for all i, k, meaning that all items have the same release time and deadline and that the utility values sum. An optimal solution to this problem instance is then a schedule that maximizes the sum of the item weights while obeying the release time and deadline constraints. Since the release times and deadlines are each all identical, ordering is irrelevant, and so an optimal schedule will consist of the items appearing in an optimal knapsack solution. □

4. ALGORITHMS

In this section, we first present two heuristics for the general case. Then we discuss a special cases to which more efficient algorithms apply.

4.1 Heuristics for the general case

First we attempt to adapt algorithms from additive utility case in this new setting.

The Two-Phase algorithm[8] first pushes job instances onto a stack when a later job's weight is greater than those of conflicting jobs already on the stack; then it pops job instances from the stack and places them in a non-overlapping schedule. This works with independent job weights because when a job is pushed onto the stack, the actual value pushed is the weight difference between that job and the overlapping jobs lower on the stack. In the more general setting, the marginal value of a job depends on which other jobs have been scheduled, which isn't determined until the second phase of the algorithm. The LP rounding method by Bar-Noy et al.[7] does not apply because of the nonlinear objective function implies a nonlinear program.

We therefore turn to adapting the combinatorial algorithm Admission[7] that considers jobs instances in the order of nondecreasing end times. If the currently considered job overlaps with a set of scheduled jobs, we compare its weight with the sum of the overlapping jobs and decide whether to replace them with it. Crucially, the weight used for the currently considered job is its marginal weight when combined with the already-scheduled *non-overlapping* jobs (see Algorithm 1 for details).

Algorithm 1 Modified Admission

```
1: $A \leftarrow \emptyset$, is the scheduled data set
2: $I \leftarrow$ the set of all unscheduled data instances
3: while $I$ is not empty do
4:   let $I_i \in I$ be an instance of $D_i$ that terminates earliest in $I$
5:   $I \leftarrow I \setminus \{I_i\}$
6:   let $C_i$ be the set of jobs in $A$ overlapping with $I_i$
7:   let $U_c \leftarrow U(A) - U(A \setminus C_i)$ and $U_i \leftarrow U(A \setminus C_i \cup \{D_i\}) - U(A \setminus C_i)$
8:   if $U_i > \beta U_c$ then
9:     $A \leftarrow A \cup \{D_i\} \setminus C_i$
10:  end if
11: end while
12: return $A$
```

The first algorithm works on data instances, considering every potential starting time of a job. However, its main focus is not on the marginal utility but instead chooses jobs in order of earliest possible finishing time. Therefore we propose a second algorithm (Algorithm 2) that focuses more on data items' marginal utility. It greedily schedules jobs with the highest marginal utility over processing time ratio. Each time an unscheduled job with the maximum ratio is considered: if it is schedulable, we schedule it at the earliest time.

4.2 Special case with $r_i = 0, d_i = d$

One special case is that all users are moving together or stay in the communication range of the base station during the same period. Within the common release time and deadline, the order of scheduled data items is irrelevant to the total utility. For each data bundle, if we consider every possible combination of data items to schedule as a "choice", this problem is exactly the Multiple Choice Knapsack Problem. Therefore, this special case can be solved by a dynamic programming algorithm[15] in pseudo-polynomial time.

Algorithm 2 Greedy Ratio

```
1: $A \leftarrow \emptyset$, is the scheduled data set
2: $N \leftarrow$ the set of all unscheduled data
3: while $N$ is not empty do
4:   let $D_i \in N$ be a job instance maximizing $\frac{U(A \cup \{D_i\}) - U(A)}{p_i}$
5:   $N \leftarrow N \setminus \{D_i\}$
6:   if $D_i$ is schedulable then
7:     schedule $D_i$ at the earliest possible time
8:     $A \leftarrow A \cup \{D_i\}$
9:   end if
10: end while
11: return $A$
```

5. EVALUATION

In this section, we evaluate two algorithms with synthetically generated problem instances.

We use the discrete time-simulation environment *REPAST Simphony*[*] to simulate a base station and mobile users in the field. In our simulation setup, we deploy the user locations uniformly at random in a 2D grid of 500m×120m, and we place the base station at the center of the field. Assuming uniform transmission power and rate between the base station and users, the transmission range is fixed at $R = 60m$. We set each user's direction of movement to be towards the base station, so that if a user is due east of the base station it will be moving west. This constrains each user to cross into the base station's transmission range exactly once over the course of the simulation. The simulation ends when all the user have passed in and then out of base station range. The times at which these events occur become the release times and deadlines. A sample scenario is shown in Figure~fig:simscenario, in which the red triangle is the base station and blue circles are the users. The red line connecting a user to the base station indicates that that user is currently within transmission range.

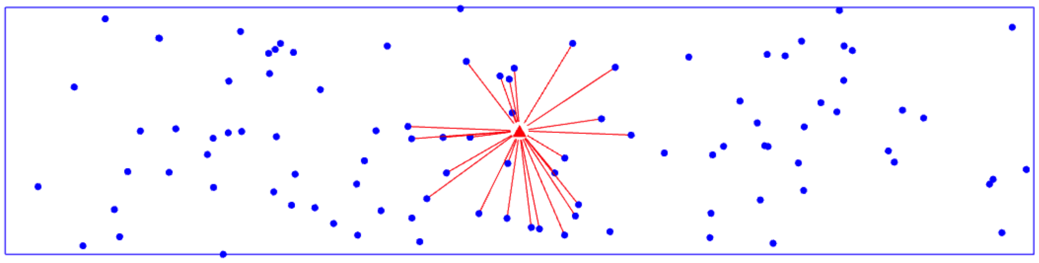


Figure 2: A sample simulation scenario.

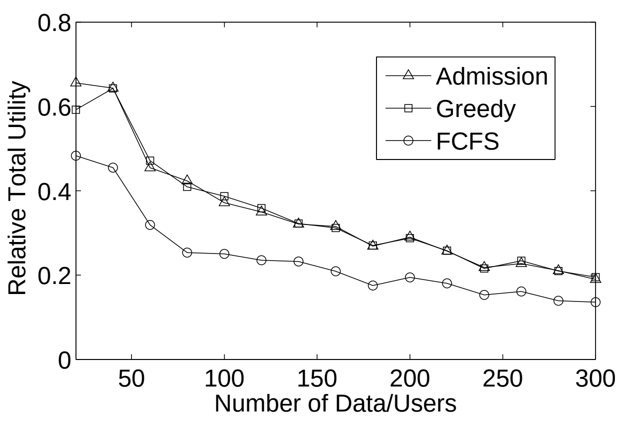
For each time step, we use Random Way Point[16] model to generate user movements. Users' speeds are randomly selected between 0 and *maxSpeed*. Each user has only one download request; multiple requests from a user can be realized as multiple users moving together. Data weights are chosen from an exponential distribution with parameter λ while processing times are chosen uniformly in the range of 0.1 to $maxPT$ times the time window size. $DataPerSet$ indicates the number of data items in a bundle, each of which is associated with either a super-additive or a sub-additive utility function. We randomly assign each data item to these $numUser/DataPerSet$ sets.

In one series of tests, we vary one of following parameters while holding the others fixed: 1. $numUser$, 2. $maxSpeed$, 3. $maxPT$, 4. $DataPerSet$. The default values of these parameters are listed in Table 2.

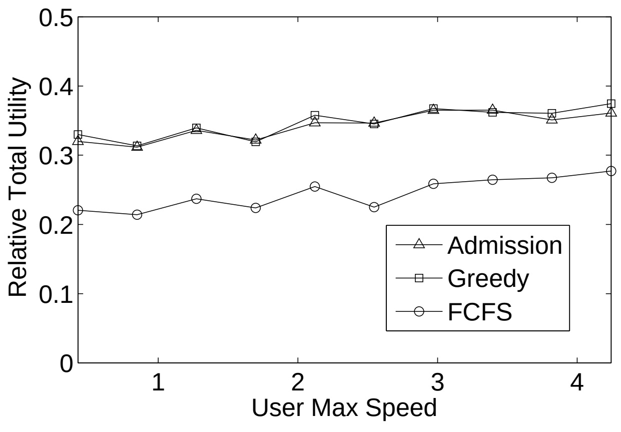
[*]http://repast.sourceforge.net

Table 2: Default parameter values.

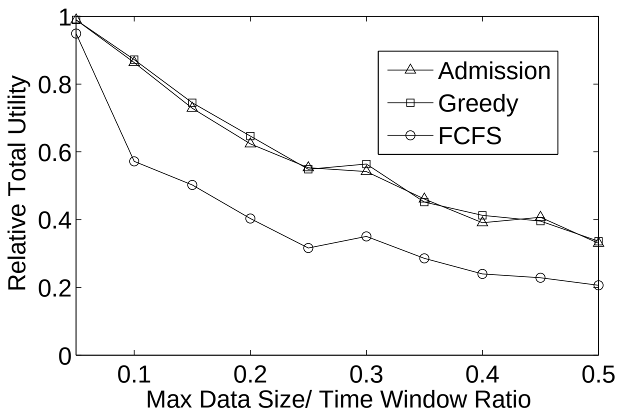
$numUser$	$maxSpeed$	$maxPT$	$DataPerSet$
100	7	0.5	5



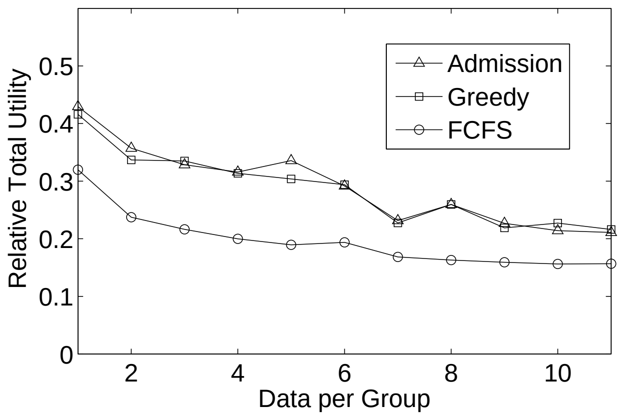
(a) Varying $numUser$.



(b) Varying $maxSpeed$.



(c) Varying $maxPT$.



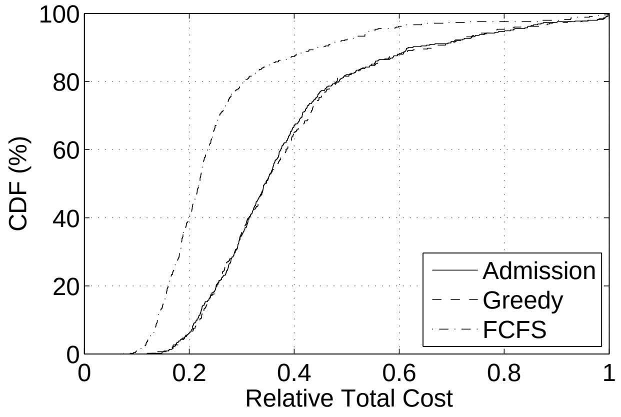
(d) Varying $DataPerSet$.

Figure 3: Simulation results.

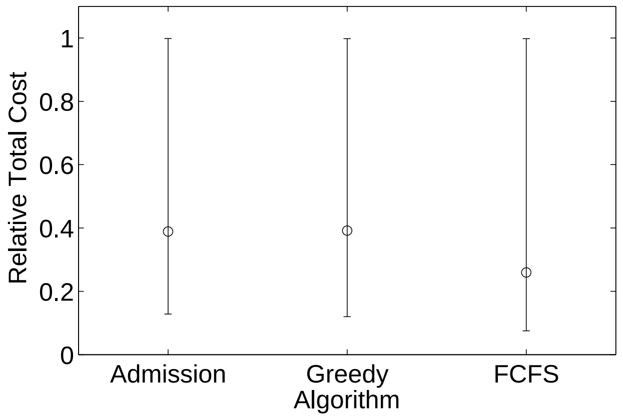
We also implement a First Come First Serve (FCFS) scheduler as baseline algorithm for comparison. For each parameter value, we generate 10 problem instances and record the total utility scheduled by all algorithms. For each problem instance, we divide the total utility value by the maximum utility of all data items, which is a (loose) upper bound of the optimal solution value. Therefore, the relative utility we report is a conservative estimate of the approximation factor obtained in our experiments. To examine the effect of different input parameters, we plot separate results of the four series of tests in Figure 3.

First, we vary the number of users from 30 to 300. As shown in Figure 3a, all three algorithms have a decrease of relative utility, which suggests that more users make the problem harder to solve. Also, when the number of users increases beyond a certain point, a limit on the total utility achievable may be reached due to the limited bandwidth.

Second, we vary $maxSpeed$ from 0.43 to 4.3 meters per timestep. As we can see in Figure 3b, as the speed increases, the relative utility increases as well. A lower speed yields a larger time window, but the maximum processing time is fixed at half the size of the time window thus it increases correspondingly. In fact, decreasing



(a) Varying $numUser$.



(b) Varying $maxSpeed$.

Figure 4: Algorithm statistics.

the speed will increase the average number of simultaneous data items within the base station's transmission range, which can make the problem more difficult.

Third, we vary the maximum data processing time. To ensure each data is schedulable, $maxPT$ must be bounded by the size of each time window. We therefore vary it from a fraction of 0.1 to 0.5 of the size of the time window. As we can see from Figure 3c, the total utility decreases very quickly. This is because a larger $maxPT$ implies that each data may occupy the channel longer, and thus it effectively reduces the number of data items that can be scheduled.

In the last series of simulations, we vary the number of data items per set from 1 to 12. When there is only one data per set, the problem collapses to the additive utility setting. As we can see from Figure 3d, that setting is the easiest case to solve for all three algorithms. When we increase the number of data items per set, the impact of nonadditive utility grows and the relative utility decreases.

Figure 4 presents some statistics on the overall performance. As shown in the CDF of relative utility (Figure 4a), the two algorithms perform much better than the baseline FCFS algorithm does, with Greedy Ratio doing slightly better than Admission. Although 80% of the tests achieve relative utility less than 0.5, this may be due to the looseness on the optimal value bound used. We also plot an error bar in Figure 4b using the max, min and median of relative costs for each algorithm in order to investigate their stability. Admission is the most stable but with only a slight advantage over Greedy Ratio, which in turn has the highest median utility value. Overall, it appears that Greedy Ratio may have the most to recommend, since operating on data items rather than data item instances makes it quite efficient.

6. CONCLUSION

In this paper we studied the broadcast scheduling problem under a new setting in which the joint utility of data is in general different from the sum of the individual utilities. We began by formulating the problem and proving its hardness. Then we adapted an existing algorithm for the additive utility case and proposed a new greedy algorithm. Using synthetically generated problem instances, we evaluated the two algorithms. As well as algorithms for the general case, we analyzed a special case in which data have a common time window and found that it can solved optimally by dynamic programming.

In the future, we plan to look at more special cases which may admit efficient optimal algorithms, such as the case with $p_i = 1$ and sub-modular joint utility. We will implement the formulation with optimization software to obtain the optimal, in order to better evaluate the algorithms. We also plan to adapt the data bundles concept to more complex scenarios such as convergecast.

REFERENCES

1. F. Bass, E. Pessemier, and D. Tigert, “Complementary and substitute patterns of purchasing and use,” *Journal of Advertising Research* **9**(2), pp. 19–27, 1969.
2. J. Jing, A. S. Helal, and A. Elmagarmid, “Client-server computing in mobile environments,” *ACM Computing Surveys* **31**(2), 1999.
3. U. Lee, E. Magistretti, M. Gerla, P. Bellavista, and A. Corradi, “Dissemination and Harvesting of Urban Data Using Vehicular Sensing Platforms,” *IEEE transactions on vehicular technology* **58**(2), pp. 882–901, 2009.
4. A. Balasubramanian, B. Levine, and A. Venkataramani, “Enhancing interactive web applications in hybrid networks,” in *Proceedings of the 14th ACM international conference on Mobile computing and networking*, pp. 70–80, ACM, 2008.
5. S. Jiang and N. Vaidya, “Scheduling data broadcast to impatient users,” in *Proceedings of the 1st ACM international workshop on Data engineering for wireless and mobile access*, p. 59, ACM, 1999.
6. J. Xu, X. Tang, and W. Lee, “Time-critical on-demand data broadcast: Algorithms, analysis, and performance evaluation,” *IEEE Transactions on Parallel and Distributed Systems* , pp. 3–14, 2006.
7. A. Bar-Noy, S. Guha, J. Naor, and B. Schieber, “Approximating the throughput of multiple machines in real-time scheduling,” *SIAM Journal of Computing* **31**(2), 2001.
8. P. Berman and B. DasGupta, “Multi-phase algorithms for throughput maximization for real-time scheduling,” *Journal of Combinatorial Optimization* **4**(3), 2000.
9. A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber, “A unified approach to approximating resource allocation and scheduling,” *J. ACM* **48**(5), pp. 1069–1090, 2001.
10. H. Rowaihy, M. P. Johnson, D. Pizzocaro, A. Bar-Noy, L. Kaplan, T. L. Porta, and A. Preece, “Detection and localization sensor assignment with exact and fuzzy locations,” in *DCOSS’09, Marina Del Rey, California, USA*, pp. 28–43, June 2009.
11. A. Kelly, “Precision dilution in triangulation-based mobile robot position estimation,” in *Proceedings of Intelligent Autonomous Systems*, (Amsterdam), 2003.
12. J. Abrache, T. G. Crainic, M. Gendreau, and M. Rekik, “Combinatorial auctions,” *Annals of Operations Research* **153**(1), pp. 131–164, 2007.
13. M. H. Rothkopf, A. Peke?, and R. M. Harstad, “Computationally manageable combinational auctions,” *Management Science* **44**, pp. 1131–1147, Aug. 1998. ArticleType: primary_article / Full publication date: Aug., 1998 / Copyright 1998 INFORMS.
14. S. de Vries and R. Vohra, “Combinatorial auctions: a survey,” *INFORMS J. on Computing* **15-3**, pp. 284–309, 2003.
15. K. Dudziski and S. Walukiewicz, “Exact methods for the knapsack problem and its generalizations,” *European Journal of Operational Research* **28**(1), pp. 3–21, 1987.
16. J. Broch, D. A. Maltz, D. B. Johnson, Y. C. Hu, and J. Jetcheva, “A performance comparison of multi-hop wireless ad hoc network routing protocols,” in *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pp. 85–97, ACM New York, NY, USA, 1998.