Personal and Ubiquitous Computing (2022) 26:781–794
https://doi.org/10.1007/s00779-020-01423-1

ORIGINAL ARTICLE

Set it and forget it: utility-based scheduling for public displays

Kristi Bushman[1] · Alexandros Labrinidis[1]

Received: 16 August 2019 / Accepted: 3 June 2020 / Published online: 20 June 2020

Abstract

The pervasiveness of public displays is prompting an increased need for "fresh" content to be shown, that is highly engaging and useful to passerbys. As such, live or time-sensitive content is often shown in conjunction with "traditional" static content, which creates scheduling challenges. In this work, we propose a utility-based framework that can be used to represent the usefulness of a content item over time. We develop a novel scheduling algorithm for handling live and non-live content on public displays using our utility-based framework. We experimentally evaluate our proposed algorithm against a number of alternatives under a variety of workloads; the results show that our algorithm performs well on the proposed metrics. Additional experimental evaluation shows that our utility-based framework can handle changes in priorities and deadlines of content items, without requiring any involvement by the display owner beyond the initial setup.

Keywords Pervasive displays · Scheduling algorithm · Utility function · Deadlines

1 Introduction

Pervasive displays are becoming a regular fixture of everyday city life [2]. Although the majority of such displays are still showing mostly static content, e.g., advertisements, the push and the demand for data-rich content is very high. Data-rich content is often *live* (e.g., real-time transit information[1]) or *time-sensitive* (e.g., weather forecasts). One way to address the idiosyncrasies of live content is to assign *deadlines* to it, i.e., a specific time point by which the content item should be displayed in order to have positive "value" to passerbys. Of course, such deadline-driven content may coexist with content that does not have such specific timing requirements.

[1] https://transitscreen.com

✉ Kristi Bushman
k.bushman@pitt.edu

Alexandros Labrinidis
labrinid@cs.pitt.edu

[1] Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260-9161, USA

1.1 Motivating example

Our motivating example is a public display at a bus stop that shows various content items including real-time bus arrival information, real-time traffic information, up to the minute weather information, the Twitter feed of the bus company, and advertisements. The motivation behind these content choices is to make the display "interesting," so that it does not get ignored like banner ads on web sites. Along those lines, we envision different content items being shown at separate times on the display, instead of trying to squeeze too many things in a single screen at the same time.

Given this setup, we want to determine the best schedule to show the various content items. Clearly, the different types of content have different "value" to the people at the bus stop and that value changes over time. For example, it is absolutely crucial that an alert about a bus arrival be shown shortly before the bus arrives (30 s–1 min) and definitely not after the bus leaves the bus stop. The exact arrival time of the bus (i.e., the "deadline") is often not the originally scheduled time, since it is affected by current traffic conditions, and therefore not known well in advance. Additionally, other content types (e.g., Twitter feeds) do not have such strict timing constraints, but must also be shown.

Although bus alerts are high priority, we would like to integrate them into the schedule in a way that feels natural. Traditional scheduling methods have handled high priority content items by allowing them to preempt less important items. However, given that we would be showing these alerts hundreds of times a day, we would prefer to integrate the alerts into the schedule in a less intrusive manner.

1.2 Requirements of an ideal scheduling algorithm

Given the motivating example, we would like to have a scheduling algorithm with the following characteristics:

- Can handle scheduling constraints (e.g., what time or how often to show particular items).
- Can deal with content items being added to or deleted from the list of available content (even without significant advance notice).
- Can handle content that is deadline-driven, but also content that is not.
- Can consider the different "value" content has to viewers and use it to prioritize scheduling decisions.
- Can adapt the schedule in response to new priorities or information.
- Can integrate time-sensitive content in a non-preemptive manner.

To the best of our knowledge, there is no scheduling algorithm that addresses all of the above characteristics.

1.3 Contributions of this work

We make the following contributions:

1. We propose a *static utility function framework* to capture the inherent "value" of both deadline-driven and traditional content items.
2. We extend the static utility function framework to a *self-adjusting utility function framework* that is more user-friendly and can adapt to changing priorities and deadlines.
3. We develop a *novel scheduling algorithm* (Lookahead algorithm) that uses the utility function framework to minimize missed deadlines while maximizing the overall utility of content items shown.

2 Related work

Most commercial digital signage players[2] allow the display owner to create playlists (an ordering of content items that play in a loop) or to specify the exact timings to show each item. With these scheduling methods, all schedule changes must be made manually by the display owner ahead of time. They cannot happen automatically in response to certain information or events.

The scope of this work is in the area of context-aware scheduling [16]. With context-aware scheduling, the scheduler decides which item to show based on the greater context that surrounds the display. Context information may be provided to the scheduler via cameras, sensors, or external information sources.

Elhart et al. [3] describe some of the key challenges that arise in context-aware scheduling. The challenges most relevant to this work include:

- introduction of new content items or scheduling constraints in real-time,
- handling constraints involving both absolute and relative timings,
- defining appropriate preemptive or priority-based behavior,
- optimizing multiple scheduling objectives, and
- providing a trade-off for competing objectives.

Some approaches in the area of context-aware scheduling are tailored for a specific display with specific scheduling criteria [9–12, 15]. Others try to provide a more general framework that can be used to implement more specialized schedulers [2, 14].

The Yarely player, designed by Clinch et al. [1], creates a playlist from the items and constraints described in a Content Descriptor Set. The playlist is cycled through on the player in a round-robin fashion. High-priority content, such as emergency alerts or personalized content items, are able to interrupt the playlist.

Ribeiro and José [13] created a model to describe the timeliness for two different categories of time-sensitive content: information items and event-related items. The timeliness of an information item decreases as time elapses since the publication date. An event-related item becomes more timely as the event gets closer, then loses timeliness when the event is over. The scheduler chooses from among the most timely items to decide what to display on the screen.

Elhart et al. [4] designed a framework and API for scheduling both interactive and non-interactive applications. The API allows applications to request display resources in response to certain information or events. Interactive content is scheduled by preempting non-interactive content.

Mikusz et al. [8] show how a lottery-based approach can be used to meet the requirements of several different scheduling policies. The lottery scheduling algorithm allocates tickets to content items based on some scheduling policy, then randomly draws a ticket to decide which item

[2]For example, https://screen.cloud

to show. A change in scheduling priorities can be reflected by changing the allocation of lottery tickets.

3 Scheduler architecture

We envision a public display scheduler that consists of three core components: (a) *a content library*, (b) *a filterer*, and (c) *a content scheduler*. A diagram is shown in Fig. 1.

The content library stores information about the applications, as well as the metadata needed for scheduling purposes. Content items can be added to or removed from the library at any time. A content item must include the following: application info (image, video, web URL, etc.), duration, valid days and times, and a utility function which describes the item's value over time. The content library has a sub-component called the *utility function adjuster* which can be used to alter the utility function of an item. Utility functions are discussed in detail in Section 4.

The filterer pulls content items from the library and removes any invalid items before passing them to the scheduler. A content item is invalid if the current time is not within valid times specified for the item.

From the items that make it through the filterer, the content scheduler decides which item to show next. The scheduler uses the Lookahead algorithm, which makes scheduling decisions based on the utility function of each item. This algorithm is discussed in detail in Section 5.

All interactions between the display owner and the system occur through an API to the content library. The display owner does not interact with the scheduler directly, but can instead make scheduling changes by changing the metadata of the content items. The API allows content items to be added, deleted, or updated.

4 Utility functions for content items

Utility functions (UF's) are used in many disciplines in order to express value over time. UF's that express the value of job completion over time have been used for scheduling tasks in real-time operating systems [5], database systems [6], and HPC systems [7].

We propose using a utility function to represent the viewer-perceived value of showing a content item over time. These utility functions can then be used to inform scheduling decisions.

4.1 Static utility function framework

Our framework supports two types of content: (a) *anytime content (AC)* and (b) *deadline-driven content (DC)*. We believe these two categories encapsulate many different content items, however, certain items such as interactive items may not fit well into the framework.

Most items traditionally shown on public displays are anytime content. This type of content has no inherent value tied to a specific time of day. However, it may increase in value to viewers if not shown for some period of time. An example of an AC item is a weather application. The weather is valuable to viewers at any time of day, however, it is not valuable to show the weather twice in 1 min, as major updates to the forecast are unlikely. Other examples of AC items include news applications, Twitter feeds, and advertisements. Typically, these items are shown multiple times throughout the day.

A deadline-driven content item is tied to a very specific time of day. These items can be thought of as "pseudo-interrupt" content, because they require immediate attention at a specific time. However, this type of content differs from true interrupt-style content (e.g., emergency alerts): in our case, we are aware of the content item in advance of when it should be shown.

Often, these DC items will be related to live events. An example of a DC item is an alert that says "Bus Arriving Now." Ideally, this item would be shown 30 s before the bus arrives. Showing this item too early or too late would cause confusion and provide no value to viewers. Other examples of DC items include event reminders and live video streams. DC items are only shown one time throughout the day.

The utility function of an AC item is a non-decreasing function, where the x-axis units are time offsets relative to the time when the content item was last shown. The y-axis units are the viewer-perceived value of showing the item. Immediately after an AC item is shown, the value of its utility function goes back to the value at time offset zero. The utility function of a DC item must increase from zero at some time (t_s) and return to zero at its deadline (t_d). The x-axis units for a DC utility function are absolute times of day. The y-axis units are viewer-perceived value. For both AC and DC items, the utility acquired by showing the item at a particular time is represented by the integral of the UF over the duration (d) that the content item is shown.

Using this framework, the display owner must design a utility function for each content item that they wish to add to the content library. The API is used to add and remove content items. The display owner can write a script to make API calls at certain times or in response to certain events so that the display can contain time-sensitive items.

4.2 Self-adjusting utility functions

The utility adjuster component can be added to the content library to allow a display owner to take advantage of self-adjusting utility functions. A self-adjusting utility function is an extension of the static utility function. This extension serves two main purposes. First, it allows the utility functions to dynamically change over time in response to new information or changing priorities. Second, it allows for much of the burden of utility function creation to be removed from the display owner. Designing static utility functions that will result in a high quality schedule can be difficult. It may require much trial and error before static utility functions behave and interact as expected. The self-adjusting utility function framework allows the display owner to specify rules about how their ideal schedule behaves, then the utility adjuster component will help to create utility functions to satisfy the rules, and update them over time as needed.

4.2.1 Self-adjusting DC functions

When using self-adjusting utility functions, the shape of each deadline-driven utility function must still be specified up front by the display owner. However, instead of describing the shape in terms of absolute time, the shape is specified using time points relative to the deadline. Now, if the deadline is changed, the utility function can be redrawn with respect to the new estimate of the deadline. For cases such as the motivating example, where the deadlines are not known well in advance, the estimate of each deadline may change over time. In order to use a self-adjusting DC function, the display owner would write a script that monitors the deadline (in the case of the bus alerts, the script would monitor the expected bus arrival time) and make an API call to change the deadline of the utility function whenever appropriate.

4.2.2 Self-adjusting AC functions

For anytime content items, rather than designing a utility function, the display owner can specify rules that prescribe the amount of playtime that each item should have. For example, “Play item A for 30% of the available playtime.” The available playtime is defined as any playtime not taken up by deadline-driven items (since those items should have priority). The display owner can write a script that makes API calls to tell the utility adjuster component to change the playtime percentage at certain times or in response to certain events. Alternatively, these percentages could be set through a user interface.

When the display owner specifies playtime rules in this manner, the utility adjuster component will create utility functions to be used in the system; however, these are abstracted away from the display owner. Each AC item will begin with the exact same shape utility function, which will dynamically change based on the display owner’s rules. Each of these AC functions has a *scale factor*, which will *stretch* or *compress* the x-axis of the utility function, causing the item to take more or less time in order to reach high utility (Fig. 2). The faster an item reaches high utility, the more often it will be shown. Each scale factor is initially set to 1. According to the rules specified by the display owner, the scale factors for each item will increase or decrease, modifying the utility functions until they result in a schedule such that each of the playtime rules are satisfied.

To adjust the scale factors of the AC utility functions, the utility function adjuster component analyzes the content chosen to play during the previous h minutes of history from the schedule. If an item was played significantly less than its target playtime during this section of history, the scale factor is decreased by p percent (compressing the utility function in the x direction). If the item was played significantly more than its target playtime, the scale factor is increased by p percent (stretching the utility function in the x direction). If the historical percentage is within 2% of the target percentage, the scale factor does not change. This process is repeated every h minute.

The parameters h and p affect how quickly the schedule is able to converge to the rules and how stable the playtime percentages are over time. Sensitivity analysis on these parameters is shown in Section 7.4.

This manner of scheduling using self-adjusting AC functions can be generalized for use in traditional public display applications that do not include any deadline-driven content.

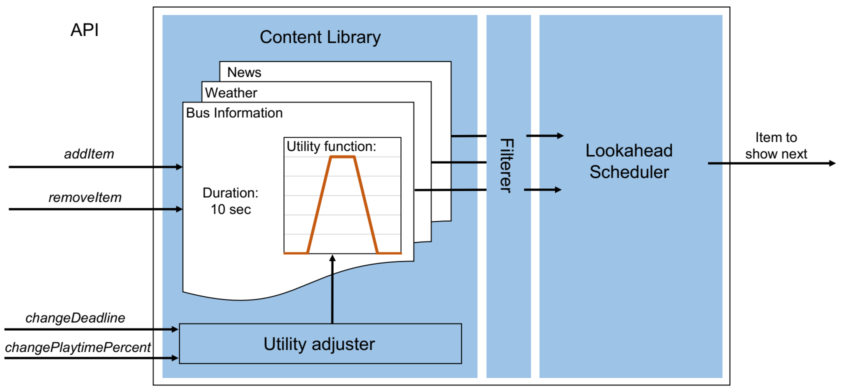


Fig. 1 Scheduler architecture

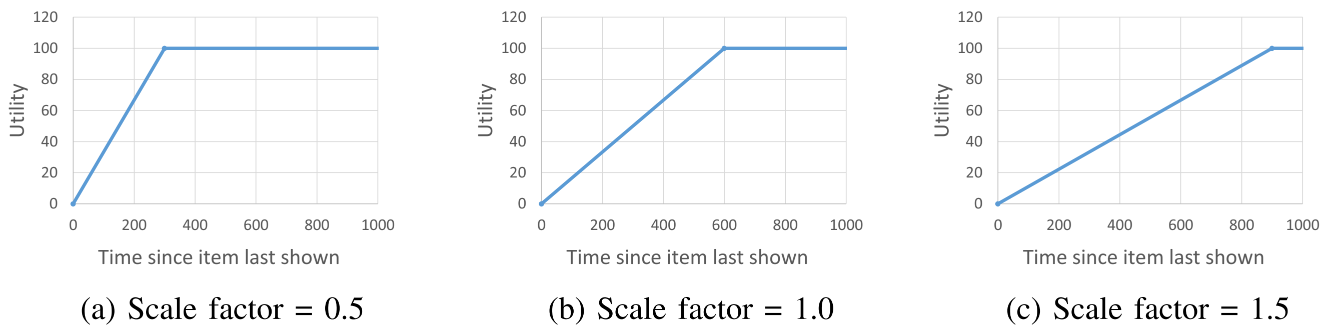


Fig. 2 Scale factors stretch or compress the AC utility function in the *x* direction. When the utility function is compressed, the item acquires higher utility sooner. When the utility function is stretched, it takes longer for the item to acquire high utility. **a** Scale factor = 0.5. **b** Scale factor = 1.0. **c** Scale factor = 1.5

5 Lookahead scheduling algorithm

In this section, we propose an algorithm called the Lookahead algorithm (LA) for scheduling content on public displays using our utility function framework. In short, the algorithm looks at when deadline-driven content items are going to require airtime, then schedules anytime content items around the DC items so that no preemptions are necessary. The goal is to decide which content item to show next in order to maximize both the total utility of content shown and the number of DC items that are shown before their deadlines.

Note that the scheduler only sees a static representation of the utility function. If self-adjusting utility functions are being used, the scheduler sees the current state of the utility function as if it were static.

To decide which content item to show next, at time t_n, we construct a lookahead window of size w (seconds). The lookahead window is a period of time where we will build a hypothetical schedule of items that would ideally be shown in the near future. This window helps inform our decision of which content item to show at time t_n. When constructing this hypothetical schedule, we first only consider DC UF's that are valuable within the window. We will decide whether to show an AC item after construction of the hypothetical schedule is completed.

For each DC item, we calculate the slack of its utility function. Our definition of slack was inspired by the notion of slack in operating systems.[3] Slack measures how many time slots are available for scheduling the content item while also receiving positive utility value. Larger slack means there are more options for when to schedule that item. The formula for slack is as follows:

$$slack = \frac{t_d - \max(t_n, t_s)}{d} \tag{1}$$

[3]https://wikipedia.org/wiki/Least_slack_time_scheduling

We place DC items on the hypothetical schedule in order of increasing slack (i.e., DC items with the least slack are placed first). Each content item is placed at the time where its acquired utility (integral of the UF) is maximized given that it does not conflict with any item already on the schedule. If there are multiple time slots that tie for the highest acquired utility, the item is placed in the earliest of those time slots. A content item is not placed on the hypothetical schedule if it cannot acquire positive utility.

$$slack = \frac{t_d - \max(t_n, t_s)}{d} \tag{2}$$

Once all valid DC items have been placed on the hypothetical schedule, we look at the very beginning of the hypothetical schedule (at time t_n). If there is a content item placed here on the hypothetical schedule, that is the content item that will be shown next. Otherwise, we calculate the gap of time from t_n to the first DC item on the hypothetical schedule. Out of the AC items with a duration that would fit in this gap, the item with the highest utility density (3) [5] is the item that will be shown next.

$$density = \frac{\int_{t_n}^{t_n+d} UF}{d} \tag{3}$$

This decision process is executed within the last second of showing the current item so that the decision of what to show next is based on the most current knowledge of upcoming content (new content items, updated utility functions, etc.). Each time a decision is made, the hypothetical schedule is completely reconstructed. Although content items are likely to be placed in the same time slot on the hypothetical schedule for many iterations of the decision process, recomputing the hypothetical schedule with every iteration allows the algorithm to be responsive to changing content, while still using available knowledge to inform the current decision.

6 Evaluation of the Lookahead algorithm

6.1 Evaluation environment

We implemented a simulator program in Python to evaluate different scheduling algorithms; it was executed on a Dell machine with an Intel Core i7 3.4 GHz processor and 32 GB of RAM.

6.2 Algorithms evaluated

We evaluated the performance of our scheduling algorithm compared with seven different baselines. These baselines are algorithms that are commonly used in display scheduling or operating systems, but have been adapted to make sense under our utility-based framework.

– *Earliest Deadline First/Greedy (EDF/G)*: For any DC item that would acquire positive utility if shown next, show the content item with the earliest deadline. If there are no such DC items, show the AC item with the highest utility density.
– *Earliest Deadline First/Random (EDF/R)*: For any DC item that would acquire positive utility if shown next, show the content item with the earliest deadline. If there are no such DC items, show a randomly chosen AC item.
– *Greedy (G)*: Choose the content item with the highest utility density (as specified in (2)).
– *Lookahead (LA)*: As described in Section 5. A lookahead window of 5 min was used in our evaluation.
– *Lottery with Current Utility (LOT-C)*: Allocate lottery tickets based on the current height of the item's UF. Randomly draw a ticket to decide what to show next.
– *Lottery with Maximum Utility (LOT-M)*: Allocate lottery tickets based on the maximum height of an item's UF. Randomly draw a ticket to decide what to show next.
– *Random (RAND)*: Out of all content items that would acquire positive utility if shown next, randomly select which item to show.
– *Round Robin (RR)*: Show all content items in a circular order, skipping a content item if it would not acquire positive utility.

6.3 Workload generation

We evaluate the Lookahead algorithm on different workloads of static utility functions. We do not consider self-adjusting utility functions in this section since the scheduler only sees a static representation of the utility function anyway. We will demonstrate the use of self-adjusting utility functions in the next section.

We generated different workloads for our evaluation using template utility functions. These template functions were designed to be simple functions with tunable parameters that allow for the generation of workloads with different properties. The experiments in this section use the template utility functions, however, the Lookahead algorithm does not depend on these templates. In practice, the utility function for a content item can be any shape that adheres to the constraints listed in Section 4.

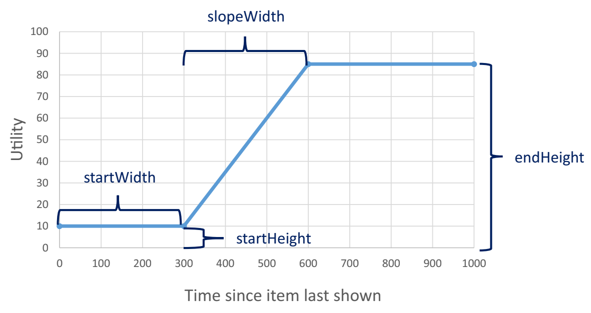
The template function for an AC item is defined by the four parameters shown in Fig. 3a. Recall that for AC items, the x-axis tracks the time passed since the content item was last shown. The intuition behind this template function is that a content item will have a value of $startHeight$ immediately after being shown. This value will be close to zero because seeing the same content item twice within a short period of time is not useful to viewers. When the content item has not been shown for $startWidth$ seconds, the value of showing the content item begins to increase. After another $slopeWidth$ seconds, the utility function reaches its maximum possible value: $endHeight$. The $endHeight$ value is indicative of the content item's overall usefulness to viewers.

The template function for a DC item is defined by the five parameters shown in Fig. 3b The intuition behind this template function is that a DC item has a $deadline$ after which, showing the item is no longer useful to viewers. The content item is useful to viewers up to $width$ seconds before the deadline. However, showing this content item would be most valuable to viewers for a period of $peakWidth$ seconds ending at $peakEnd$. The value of the utility function for this period of maximum value is $peakHeight$.

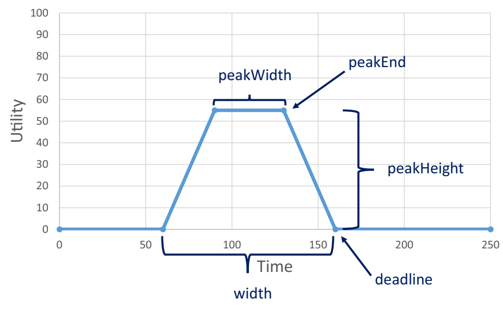
We generated a baseline workload that is realistic of our motivating example. This workload consists of 15 AC items and 288 DC items with deadlines over the course of a 24-h period (on average 1 DC item added to the content library every 5 min). The time when an item is added to the content library is its $awareTime$. At this time, the item will start to be considered by the scheduler in its decisions. In the baseline workload, all AC items are included in the content library from the beginning of the simulation. DC items are added to the content library at some time before their deadlines. The parameters used for each utility function of the baseline workload were randomly generated within the ranges shown in Table 1. In our experiments, we change certain parameters of the baseline workload to evaluate the performance of the Lookahead algorithm across a variety of workloads.

6.4 Evaluation metrics

To evaluate the performance of the scheduling algorithms, we consider two metrics: (a) *total acquired utility* and (b)



(a) AC template utility function



(b) DC template utility function

Fig. 3 Template utility functions and parameters that enable content generation for different workloads. **a** AC template utility function. **b** DC template utility function

percentage of deadlines met. Total acquired utility is the sum of the utility acquired by each content item shown over the course of the 24-h simulation (recall that utility is the integral of the utility function over the duration shown). The percentage of deadlines met is the percent of DC items that are shown and complete their full duration before their deadline. An algorithm that effectively integrates live content into the schedule would have a high total acquired utility and meet close to 100% of the deadlines.

For each of the following experiments, we measure the two metrics on different workloads and plot the result for each algorithm. An ideal algorithm should fall in the upper right corner of this plot (high utility and high deadlines met) regardless of the workload.

Table 1 Range of parameter values for utility functions of the baseline workload

UF parameter		Baseline range
DC	duration	(5, 60)
	deadline (t_d)	(1, 86400)
	width	$(duration, duration * 8)$
	awareTime	$(t_d - width - 300, t_d - width)$
	peakWidth	$(0, width)$
	peakEnd	$(t_d - width + peakWidth, t_d)$
	peakHeight	(70, 100)
AC	duration	(5, 60)
	awareTime	0
	startWidth	(0, 600)
	slopeWidth	(0, 600)
	endHeight	(40, 80)
	startHeight	$(0, endHeight)$

For each content item, the values of the parameters of the UF are chosen from a uniform distribution that spans the range listed in the table. Times and durations are shown in seconds

6.5 Notice time (Fig. 4)

Notice time refers to the amount of time before the beginning of the utility function (t_s) that a DC item is added to the content library. When an item has a longer notice time, the scheduler has more opportunity to schedule other content around it in a manner that maximizes the total utility. We evaluated the LA algorithm using workloads with long, medium, and short notice times. From the baseline workload, the range of DC *awareTime* was changed to $(deadline - width - 300, deadline - width - 240)$, $(deadline - width - 180, deadline - width - 120)$, and $(deadline - width - 60, deadline - width)$ respectively. For all three workloads, the LA algorithm outperforms the other algorithms. Even with short notice times, the LA algorithm is able to meet 96% of deadlines and acquire more utility than all of the other algorithms evaluated.

6.6 Heights of AC vs DC functions (Fig. 5)

The maximum height of the UF is an indicator of the general importance of the content item. We evaluated the LA algorithm using workloads where the heights of DC utility functions were taller, shorter, and the same height as the AC utility functions. From the baseline workload, the range of $peakHeight$ for DC items was changed to (80, 100), (20, 40), and (50, 70) respectively. The range of $endHeight$ for AC functions was changed to (20, 40), (80, 100), and (50, 70) respectively. When DC functions are taller than AC functions, the LA algorithm outperforms the other algorithms in terms of both acquired utility and deadlines met. When AC functions are taller than DC functions, the greedy algorithm acquires 7% more utility than the LA algorithm. However, it does so by not scheduling any DC items, thus meeting 99% fewer deadlines than the LA algorithm. Because the LA algorithm acquires very high utility and also integrates almost all of the live content into

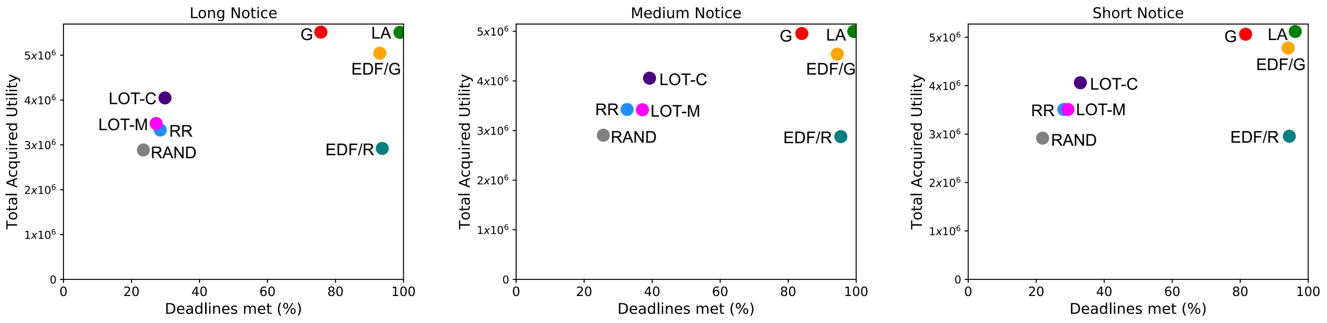


Fig. 4 Total acquired utility and percent of DC deadlines met for running scheduling algorithms on workloads with different notice times. Long notice (left), medium notice (middle), and short notice (right). The notice time refers to how far in advance the content item is added to the content library

the schedule, it is the better performing algorithm for this workload too.

6.7 Number of DC items (Figs. 6 and 7)

The number of DC items is an indication of the scheduling difficulty. The more DC items there are, the more likely it is that there are UF's that overlap in time. When utility functions overlap, it is more difficult to create a schedule such that all items meet their deadlines and acquire high utility. We evaluated the LA algorithm using workloads with low, medium, high, and very high numbers of DC items. From the baseline workload, the number of DC items was changed to 288, 576, 864, and 1440 respectively. For all four workloads, the LA algorithm is able to acquire high utility while meeting deadlines.

6.8 Number of AC items (Fig. 8)

We evaluated the LA algorithm using workloads with low (15), medium (30), and high (45) numbers of AC items. While the greedy, random, round robin, and lottery algorithms struggle to meet deadlines as the number of AC items increase, the LA algorithm is able to meet over 99% of the deadlines and acquire high utility with all three workloads.

6.9 Sensitivity of Lookahead window size (Table 2)

We evaluated the LA algorithm on the baseline workload using different lookahead window sizes. The optimal window size is dependent on the workload. The lookahead window should be at least as long as the longest duration content item. However, a slightly larger window allows the algorithm to consider more information when making a decision. The execution time of the algorithm increases linearly with the size of the window. Although, the utility acquired by the algorithm increases logarithmically with the size of the window, any window size over the longest duration content item yields reasonable performance.

6.10 Discussion

In general across all workloads, the Lookahead algorithm is the only algorithm that acquires consistently high utility and meets almost all deadlines. The random (RAND) and round robin (RR) algorithms do not perform well because they do not take into account the deadline or the utility of a

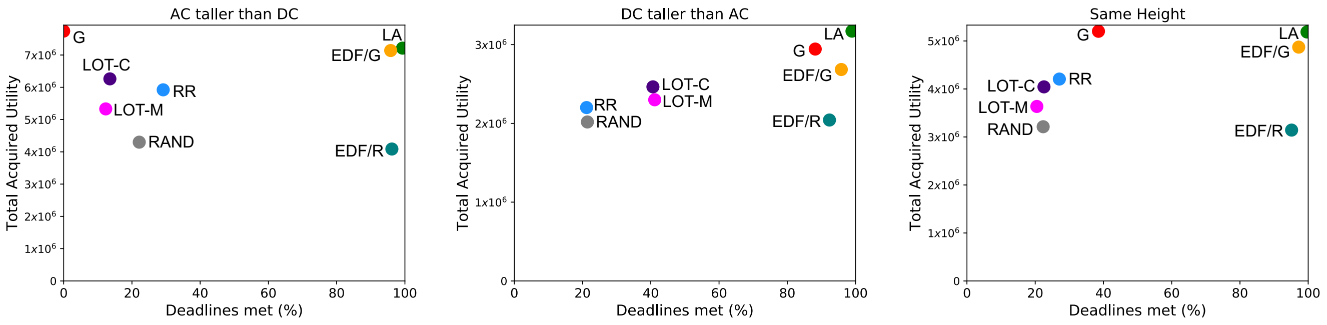


Fig. 5 Total acquired utility and percent of DC deadlines met for running scheduling algorithms on workloads with different height UF's. AC taller than DC (left), DC taller than AC (middle), and AC same height as DC (right). The max height of a utility function is indicative of the overall importance of the item to the viewer

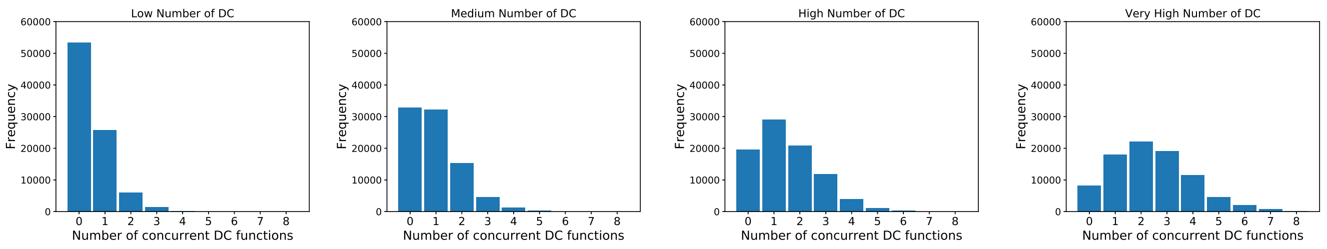


Fig. 6 Distribution of the number of overlapping DC functions throughout the 24-h period (i.e., 2 means at a given second, there are two DC functions that have non-zero value). Distributions are shown for workloads with a low number of DC content (left), medium number (center left), high number (center right), and very high number (right)

content item when making scheduling decisions. The greedy (G) and lottery-based algorithms (LOT-C, LOT-M) do not prioritize deadline-driven content so deadlines are often missed. Using an earliest deadline first protocol (EDF/G, EDF/R) can help to meet deadlines, but it is not optimal for acquiring utility. The EDF/Greedy algorithm follows a very similar procedure to the Lookahead algorithm, so it is not surprising that the two algorithms often perform similarly. However, the Looakead algorithm tends to slightly outperform the EDF/Greedy algorithm since EDF can cause deadline-driven content to be played earlier than would be optimal. These experiments have reiterated why traditional scheduling methods will not work when dealing with items whose value is dependent on time, especially when there are deadlines.

7 Evaluation of self-adjusting UFs

7.1 Simulation data

We use bus arrival data to simulate real-world examples of deadline-driven content items that would be played at a display near a bus stop. The DC items for this simulated display contain information related to the bus trips passing through the stop, so the deadlines for the content items are tied to the times when the buses arrive at the stop.

We collected bus arrival prediction data from the Port Authority of Allegheny County TrueTime API.[4] This API reports predicted arrival times for all buses arriving within the next 30 min for a given stop. We polled the API once per minute to get the latest predictions. The data used in our simulations are taken from the Forbes and Atwood stop (#29) on July 8, 2019. This stop is fairly busy, with 8 routes that pass through it.

Since the true “value” that these bus-related content items provide to viewers is based on when the items are shown with respect to the *actual* arrival time (not a prediction), the “ground truth” utility function would be based on the actual arrival time of the bus. Although we do not have data on the actual bus arrival times, we will treat the very last prediction seen on the data stream as the actual arrival time since it should be the most accurate (as this prediction is collected less than a minute before the bus arrives). Note, the Lookahead algorithm has no knowledge of a ground truth utility function. We simply use this as an retrospective evaluation metric to measure how well the static and self-adjusting utility functions perform in a real-world scenario.

7.2 Static vs self-adjusting DC utility functions (Table 3)

For each bus that passes through the stop, we would like to show two different deadline-driven items on the simulated display.

- *Item A*, an alert that the bus is currently arriving, should be shown approximately 30 s before the bus arrives.
- *Item B*, information about where the bus is heading, should be shown approximately 5 min before the bus arrives.

Since the predicted bus arrival time will change over time, the estimated deadlines for showing these content items will also change over time.

Using static DC utility functions, the display owner would write a script to watch the stream of bus arrival data. When a bus is predicted to be less than 10 min away from the stop, the script makes an API call to add an instance of item B to the content library. When a bus is predicted to be less than 5 min away, it adds an instance of item A to the content library. The deadline for the utility functions of these DC items is based on the bus arrival prediction at the time when the item is first added to the content library. The utility functions are never updated once they are in the content library, even though there may be changes to the predicted arrival times on the data stream.

Using self-adjusting DC utility functions, the display owner would also write a script to watch the stream of bus arrivals data. As soon as a new bus appears on the stream (approximately 30 min before arrival), it makes an API call

[4] https://truetime.portauthority.org/bustime/home.jsp

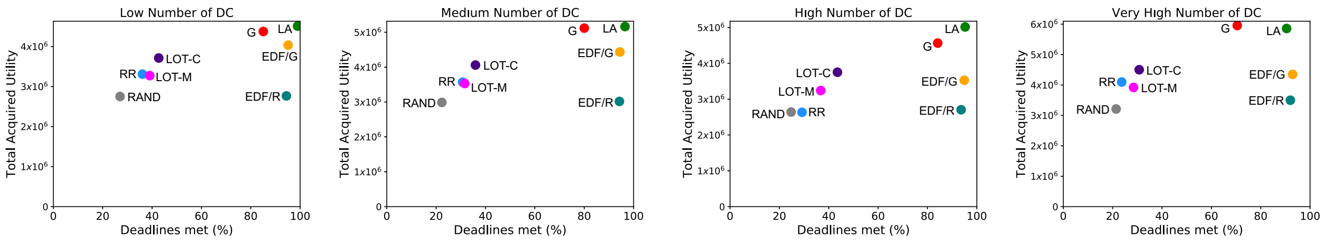


Fig. 7 Utility acquired and deadlines met for running algorithms on workloads with the UF overlap distributions shown in Fig. 6. Workloads with high numbers of DC items are more difficult to schedule because more utility functions overlap, making it more difficult to find an ordering of content that is optimal for all items

to add an instance of both item A and item B to the content library. Whenever there is a change in the predicted arrival time for this bus, the script makes an API call to update the deadlines of the utility functions.

We ran two simulations of the Lookahead algorithm, using the exact same DC items. The first simulation uses the protocol for static DC functions described above whereas the second simulation uses the protocol for self-adjusting DC functions. Both simulations also include the same four (static) AC items. After each simulation, we measured the acquired utility and percentage of deadlines met according to the "ground truth" utility functions, described in Section 7.1.

The results of these simulations are shown in Table 3. The self-adjusting utility functions were able to acquire more utility than the static functions. This increase in total utility can be attributed almost entirely to the increase in utility acquired from the DC items. Using self-adjusting utility functions, the Lookahead algorithm also meets 11.4% more deadlines than with static functions.

7.3 Utility-based vs lottery-based scheduling (Table 4, Fig. 9)

In Section 6, we showed that lottery scheduling does not work well with deadline-driven content. However, the lottery scheduling algorithm seems like an natural choice if the main scheduling constraints are target playtime percentages (as is the case when using the self-adjusting UF's for AC items). To do this, the lottery scheduler would simply allocate tickets to each item in proportion to its target playtime percentage (normalized by the duration of the content item). In this experiment, we demonstrate the benefits of our utility-based approach over the lottery-based approach for selecting AC items. For simplicity, we consider a schedule without any DC items.

We created a content set with 5 items and rules that describe their target playtime percentages. We simulated both the lottery- and utility-based approaches for selecting AC items on this content set for a 24-h period. For the self-adjusting utility functions, parameter values of $h = 5$ min and $p = 10\%$ were used. The results of this experiment are shown in Table 4.

Both approaches are able to converge close to the target playtime percentages. Although the lottery-based approach comes slightly closer to the targets, we consider this to be a negligible, unnoticeable difference to anyone observing the display. There is, however, one noticeable difference in the quality of the schedules that they create. The lottery-based approach has inconsistent spacing between instances of each content item, while the utility-based approach has much more regular spacing. This means that it is much more plausible to see a schedule such as "A A B B B A" with lottery scheduling than it would be with the utility-based approach ("A B A B A B" would be a better schedule). Since the lottery scheduling decision is based on chance, it is plausible to show the same content item multiple times in a row, or to not show a particular item at all for a very

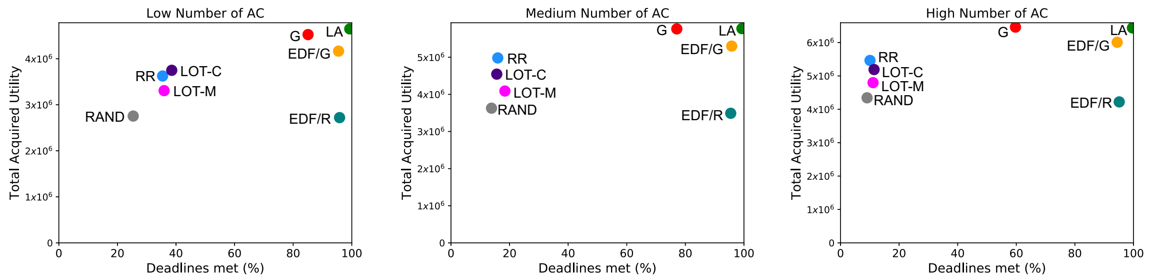


Fig. 8 Total acquired utility and percent of DC deadlines met for running scheduling algorithms on workloads with different numbers of AC items. Small number (left), medium number (middle), and high number (right)

Table 2 Performance of the Lookahead algorithm with different window sizes when the longest duration content item was 60 s

Window size (s)	Execution time (ms)	Deadlines met (%)	Total utility acquired
60	0.591	99.31	4,434,015
90	0.817	98.61	4,534,079
120	1.148	99.31	4,614,187
150	1.634	98.96	4,648,108
180	1.883	99.65	4,651,114

long time. This would not be an ideal schedule, because passerbys typically want to see a variety of content. With the Lookahead algorithm and self-adjusting utility functions, the likelihood of an item being shown is *dependent* on when it was last shown, since the utility of the item depends on the time since it was last shown. Because of this, the spacing between instances of a particular item is much more regular.

To illustrate this, we measured the spacing between instances of each content item in the schedules created by the two approaches. Spacing is defined as the number of seconds between the beginning of one instance of an item to the beginning of the next instance of the same item. The spacing distributions for each item (A through E) for the two scheduling methods are shown in Fig. 9. The distributions show that lottery scheduling creates a schedule with a wide range of spacings. Oftentimes, an item is shown twice consecutively and other times there is a long gap without showing a particular item. With utility-based scheduling, there is a much narrower distribution of spacings concentrated around some medium value. Because the utility-based approach has much more regular spacing between instances of each item, it creates a higher quality schedule.

7.4 Sensitivity of *h* and *p* (Fig. 10)

In the utility adjustment algorithm (described in Section 4.2.2), parameter h controls how often updates to the scale factors occur. Parameter p controls how drastically the utility function is stretched or compressed upon each update. Together, these parameters control how quickly the schedules are able to start adhering to the rules and how stable the playtime proportions are over time.

Table 3 Performance of the Lookahead algorithm in a real-world scenario using static vs self-adjusting DC functions

DC function type	Total utility acquired	DC utility acquired	Deadlines met (%)
Static	2,661,960	1,062,674	87.1
Self-adjusting	2,745,251	1,146,345	98.5

Table 4 Performance of the Lookahead algorithm using self-adjusting UF's vs a lottery-based approach for selecting AC items

Item	Duration	Target play time (%)	Utility play time (%)	Lottery play time (%)
A	5	5	4.1	4.8%
B	10	25	24.3	24.7
C	15	12	10.4	12.1
D	20	25	25.1	25.6
E	40	33	36.1	32.8

We ran a 24-h simulation of the Lookahead algorithm using self-adjusting AC functions with different parameters of h and p. The content items used in this experiment are shown in Table 5. Again, for simplicity, we consider a schedule with no DC items. The plots shown in Fig. 10 show the proportion of playtime that each item makes up for each 10-min interval throughout the simulation. In general, a larger h and smaller p mean it will take more time for the scale factors to converge to the proportion rules. However, a larger h and smaller p also tend to make the proportions more stable over time.

These parameters are especially important to tune correctly if the playtime percentages will be changing throughout the day (demonstrated in Section 7.5). With schedules that do not change throughout the day, a large h and small p can be used with an initial simulation phase that is run before the live scheduling begins. In this way, the schedule will have the stability seen in bottom right plot of Fig. 10, but without the initial period where the scale factors are still tuning in order to meet the playtime percentages (first 5 h of this plot).

7.5 Advanced scheduling using self-adjusting functions (Fig. 11)

In this experiment, we demonstrate the use of all components of our system together in order to create an advanced schedule. We use the same deadline-driven content items as in Section 7.2 with self-adjusting functions. The anytime content items are the same as in Section 7.4, however, the target playtime percentages change throughout the day (10% A, 35% B, and 55% C before noon, and 10% A, 55% B, and 35% C after noon). The utility adjustment algorithm was used with parameters $h = 5$ min and $p = 10\%$.

Using the Lookahead algorithm, the deadline-driven content items are able to meet 98.2% of their deadlines. The anytime content items are able to quickly adapt to the new proportions when the rules change at noon (reaching the new proportions in approximately 30 min). This can be seen in Fig. 11.

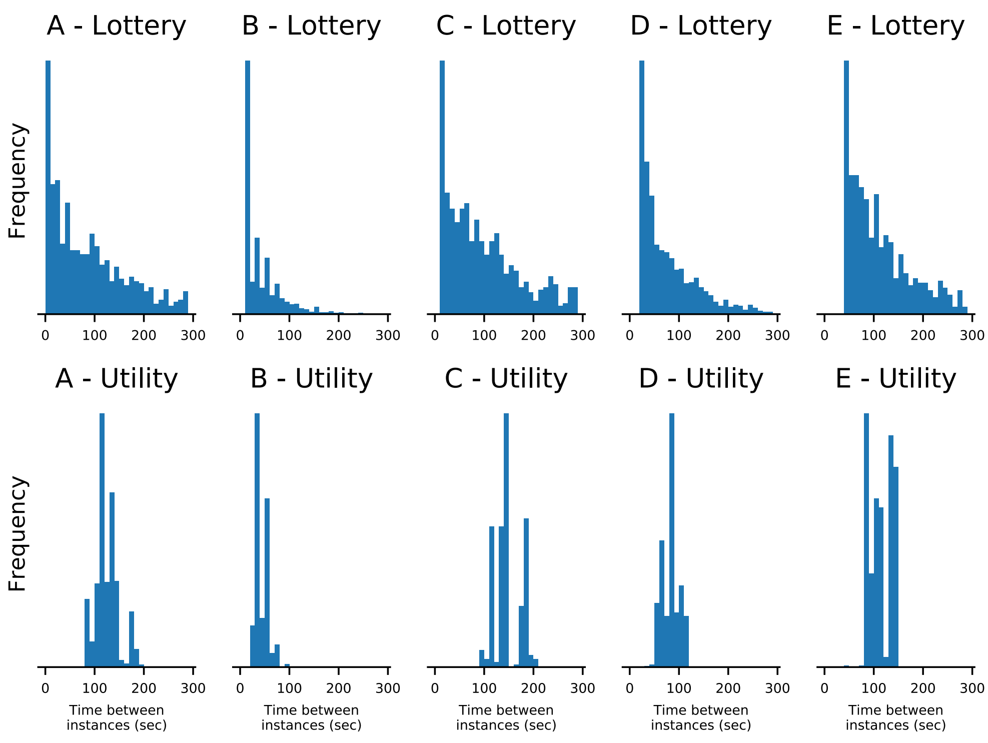


Fig. 9 Distribution of item spacings for the lottery-based approach (top) and utility-based approach (bottom) for selecting AC items. Consistent spacing between instances of an item means passerbys see a variety of content

Table 5 Items used for sensitivity analysis of h and p

Content item	Duration	Target play time (%)
A	5	10
B	20	35
C	15	55

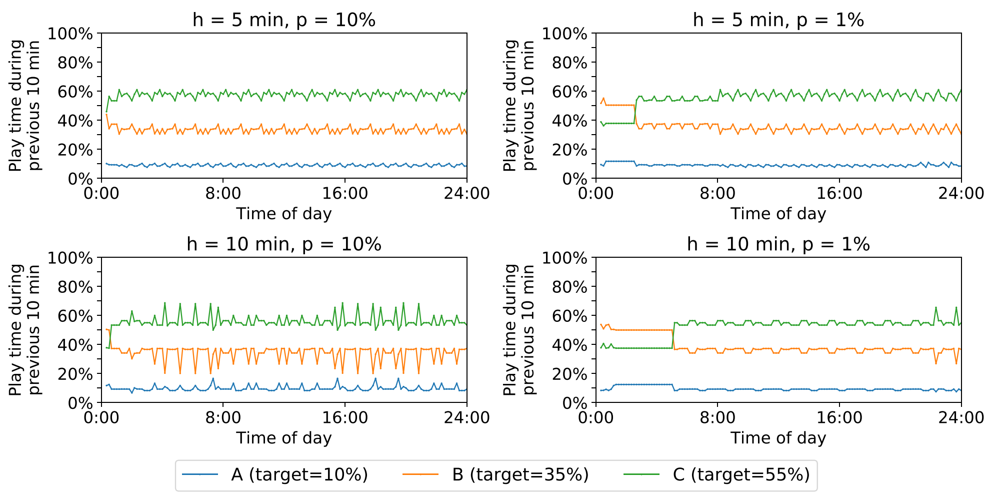


Fig. 10 Sensitivity of the utility adjustment algorithm to different values of h and p. Best viewed in color

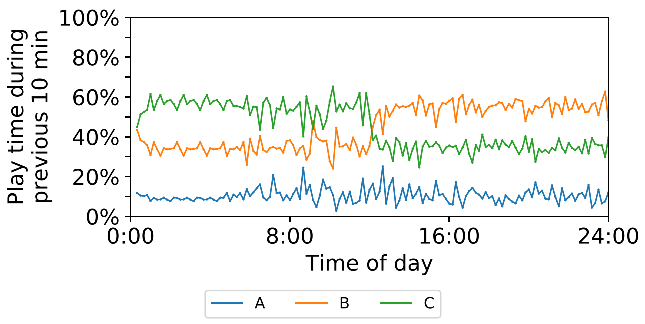


Fig. 11 Using the API to change playtime proportions throughout the day. The target proportions are 10% A, 35% B, and 55% C before noon, and 10% A, 55% B, and 35% C after noon. Best viewed in color

The ability to change the playtime proportions, can allow display owners to create schedules that accommodate time-sensitive or live content that is not necessarily deadline-driven. This feature can be used to prioritize content items based on their freshness or relevance throughout the day. For example, the proportion of weather in the schedule could increase when there is inclement weather approaching. The proportion of a particular Twitter feed could decrease when a certain hashtag is no longer trending.

8 Live deployment

We deployed a prototype of our scheduler at a public display in the lobby of an academic building. The display is similar to our motivating example, showing bus arrival information, Twitter feeds, and weather information. We used the self-adjusting utility function framework for both AC and DC items. Images of the display are shown in Fig. 12. Further work is necessary in order to make the system available to general users.

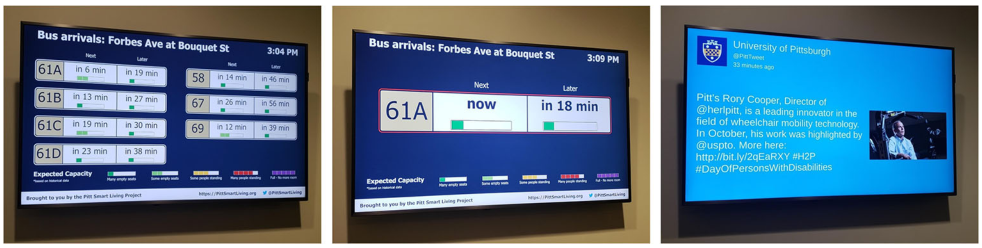
Based on our experience, we believe there is a more significant setup process for using this system compared with more traditional schedulers. The display owner must determine utility functions or rules for each item and also write a script to integrate data feeds and make API calls at the appropriate times. However, once this setup process is completed, the display owner does not have to worry about making manual changes to the schedule anymore. The schedule will contain live content that is always fresh and relevant to the viewers, with no further action required by the display owner. They can simply "set it and forget it."

9 Scope and limitations

The display at the bus stop which shows an alert for each arriving bus has been the primary inspiration for this work. Other displays that wish to show deadline-driven content may also find the work useful. We reiterate that deadline-driven content has two specific properties: (1) the content has a strict timing requirement, (2) the scheduler can be made aware of the item in advance of when the item should be shown. To the best of our knowledge, there are currently no other algorithms that can schedule this type of content in a non-preemptive manner. Although we have found the framework useful for our own proposed use case, it is currently unclear whether there is a widespread demand for content of this type in other real-world public displays.

There are also concerns about the usability of the proposed framework. Specifying a good utility function for each content item may prove to be challenging for some display owners, as this can be an abstract concept. We designed self-adjusting AC utility functions in order to remove this burden from the display owner for AC items; however, in all varieties of our framework, the display owner must specify utility functions for the deadline-driven items. We feel this is less of a concern since the notion of utility over time is much more natural for deadline-driven items given they have strict timing requirements, however, it is a concern nonetheless. An additional concern is that the display owner must have programming skills in order to use the API effectively. Given these concerns, we believe the system is not well suited for general-purpose public displays, but rather highly specialized displays that wish to show deadline-driven content. Some technical assistance may be required to set up the system.

Although the full-fledged system proposed may only be suitable for a limited number of real-life applications, certain aspects of the framework may be useful for general-purpose public displays. For instance, as shown in Section 7.3, the self-adjusting AC framework can be used regardless of the presence of DC items. Note that



Bus arrivals: Forbes Ave at Bouquet St
3:04 PM
61A
61B
61C
61D
58
67
69
Bus arrivals: Forbes Ave at Bouquet St
3:09 PM
61A
now
in 18 min
University of Pittsburgh

Fig. 12 A deployment of our prototype system. Left: a page showing bus arrival information. Center: an alert that a bus is arriving soon. Right: a Twitter feed page

without any DC items to consider, the Lookahead scheduler reduces to a Greedy scheduler (always picking the item with the highest utility density). This then becomes a tool for creating high quality ratio-based schedules, which are a common requirement for real-world public displays [1, 8]. Using this aspect of the system alone, display owners only need to specify target ratios for each item, which could easily be done through a user interface. More technology-savvy display owners could make use of an API to make changes to the ratios in response to certain information or events (demonstrated in Section 7.5). Our proposed algorithm will automatically handle the dynamicity in that case.

10 Conclusion

In this paper, we have explored supporting deadline-driven content on public displays in addition to traditional content. Our experiments have shown that our utility-based framework and Lookahead algorithm is effective for this purpose and outperforms baselines on the proposed metrics. Further work is necessary to determine whether there is a widespread need for scheduling content of this type and whether the system is practical for a general user. This is part of our future work.

Funding information This work is part of the PittSmartLiving project which is supported by NSF award CNS-1739413.

References

1. Clinch S, Davies N, Friday A, Clinch G (2013) Yarely: a software player for open pervasive display networks. In: Proceedings of the 2nd ACM International Symposium on Pervasive Displays, PerDis '13. ACM, New York, pp 25–30. https://doi.org/10.1145/2491568.2491575
2. Davies N, Clinch S, Alt F (2014) Pervasive displays: understanding the future of digital signage. Synthesis Lect Mob Pervas Comput 8:1–128. https://doi.org/10.2200/S00558ED1V01Y201312MPC011
3. Elhart I, Langheinrich M, Davies N, José R (2013) Key challenges in application and content scheduling for open pervasive display networks. In: 2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp 393–396. https://doi.org/10.1109/PerComW.2013.6529524
4. Elhart I, Langheinrich M, Memarovic N, Heikkinen T (2014) Scheduling interactive and concurrently running applications in pervasive display networks. In: Proceedings of The International Symposium on Pervasive Displays, PerDis '14. ACM, New York, pp 104:104–104:109. https://doi.org/10.1145/2611009.2611039
5. Jensen ED, Locke CD, Tokuda H (1985) A time-driven scheduling model for real-time operating systems. In: RTSS
6. Labrinidis A, Qu H, Xu J (2007) Quality contracts for real-time enterprises. In: Lecture Notes in Computer Science 4365: Post Proceedings of First International Workshop on Business Intelligence for the Real Time Enterprise. BIRTE'06 was held in conjunction with the VLDB'06 Conference, Seoul, pp 143–156
7. Lee CB, Snavely AE (2007) Precise and realistic utility functions for user-centric performance analysis of schedulers. In: Proceedings of the 16th International Symposium on High Performance Distributed Computing, HPDC '07. ACM, New York, pp 107–116. https://doi.org/10.1145/1272366.1272381
8. Mikusz M, Clinch S, Davies N (2015) Are you feeling lucky?: Lottery-based scheduling for public displays. In: Proceedings of the 4th International Symposium on Pervasive Displays, PerDis '15. ACM, New York, pp 123–129. https://doi.org/10.1145/2757710.2757721
9. Müller J, Exeler J, Buzeck M, Krüger A (2009) Reflectivesigns: Digital signs that adapt to audience attention. In: Proceedings of the 7th International Conference on Pervasive Computing, Pervasive '09. Springer, Berlin, pp 17–24. https://doi.org/10.1007/978-3-642-01516-8_3
10. Ribeiro F, Jose R (2007) Proactive scheduling for situated displays. In: Workshop on Ambient Intelligence Technologies and Applications
11. Ribeiro F, Jose R (2010) Autonomous and context-aware scheduling for public displays using place-based tag clouds, pp 131–138. https://doi.org/10.1007/978-3-642-13268-1_16
12. Ribeiro F, Jose R (2013) Smart content selection for public displays in ambient intelligence environments. Int J Ambient Comput Intell 5:35–55. https://doi.org/10.4018/jaci.2013040103
13. Ribeiro FRSG, José R (2009) Timeliness for dynamic source selection in situated public displays. In: WEBIST
14. Storz O, Friday A, Davies N (2006) Supporting content scheduling on situated public displays. Comput Graph 30(5):681–691. https://doi.org/10.1016/j.cag.2006.07.002
15. Taniguchi Y (2018) Content scheduling and adaptation for networked and context-aware digital signage: a literature survey. ITE Trans Media Technol Appl 6(1):18–29. https://doi.org/10.3169/mta.6.18
16. Taniguchi Y, Arai H, Tsutsuguchi K, Akutsu A (2014) Content-schedule optimization of digital signage taking account of location characteristics