Simultaneous Placement and Scheduling of Sensors

Andreas Krause, Ram Rajagopal
Anupam Gupta, Carlos Guestrin

October 2008
CMU-ML-08-114

Carnegie Mellon

Simultaneous Placement and Scheduling of Sensors

Andreas Krause* **Ram Rajagopal†**
Anupam Gupta* **Carlos Guestrin***

October 2008
CMU-ML-08-114

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We consider the problem of monitoring spatial phenomena, such as road speeds on a highway, using wireless sensors with limited battery life. A central question is to decide *where* to locate these sensors to best predict the phenomenon at the unsensed locations. However, given the power constraints, we also need to determine *when* to selectively activate these sensors in order to maximize the performance while satisfying lifetime requirements. Traditionally, these two problems of sensor placement and scheduling have been considered separately from each other; one first decides where to place the sensors, and then when to activate them.

In this paper, we present an efficient algorithm, eSPASS, that simultaneously optimizes the placement and the schedule. We prove that eSPASS provides a constant-factor approximation to the optimal solution of this NP-hard optimization problem. A salient feature of our approach is that it obtains "balanced" schedules that perform uniformly well over time, rather than only on average. We then extend the algorithm to allow for a smooth power-accuracy tradeoff. Our algorithm applies to complex settings where the sensing quality of a set of sensors is measured, e.g., in the improvement of prediction accuracy (more formally, to situations where the sensing quality function is *submodular*). We present extensive empirical studies on several sensing tasks, and our results show that simultaneously placing and scheduling gives drastically improved performance compared to separate placement and scheduling (e.g., a 33% improvement in network lifetime on the traffic prediction task).

* School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
† EECS, University of California, Berkeley, CA, USA.

Keywords: Sensor networks, spatial monitoring, sensor placement, approximation algorithms

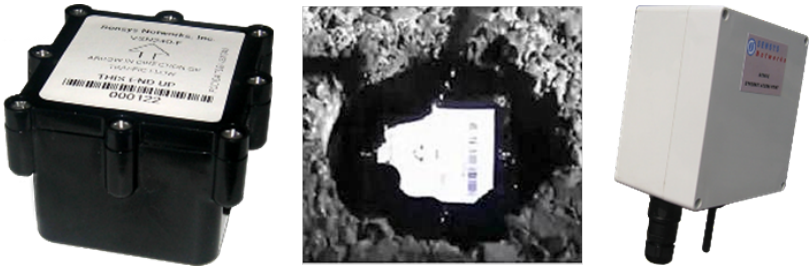


Figure 1: Sensys Networks wireless traffic sensor. (left) encased unit, (middle) sensor deployed in pavement, (right) GPRS/CDMA base station.

1 Introduction

When monitoring spatial phenomena, such as road speeds on a highway, deciding where to *place* a small number of sensors to obtain best prediction accuracy is an important task. Figure 1 shows a Sensys Networks wireless traffic sensor (Haoui et al., 2008), that provides 30 second aggregate speed, flow and vehicle density measurements. Currently the system is being deployed by Caltrans at different sites in California, including highways and arterial roads. When using such wireless sensor networks, power consumption is a key constraint, since every measurement drains the battery. For applications such as road speed monitoring, a minimum battery lifetime is required to ensure feasibility of the sensor network deployment. One approach to meeting such lifetime requirements is to deploy few nodes with large batteries. However, such an approach can be sensitive to node failures. Additionally, packaging constraints can limit the size of the battery deployed with the nodes. For these and other reasons, it can be more effective to deploy a larger number of nodes with smaller batteries, that are activated only a fraction of the time. Hence, to improve the lifetime of such a sensor network, the problem of *scheduling* becomes of crucial importance: Given a fixed placement of sensors, when should we turn each sensor on in order to obtain high monitoring performance over all time steps? One approach that has been found effective in the past is to partition the sensors into k groups (Abrams et al., 2004; Deshpande et al., '08; Koushanfary et al., 2006). By activating a different group of sensors at each time step and cyclicly shifting through these groups, the network lifetime can effectively be increased by a factor of k. In the traffic network application, current studies indicate that an increase by a factor of $k = 4$ would be required to make sensor deployment an economically feasible option (*c.f.*, Section 6 for more details).

Traditionally, sensor placement and sensor scheduling have been considered separately from each other – one first decides where to place the sensors, and then when to activate them. In this paper, we present an efficient algorithm, eSPASS (for *efficient Simultaneous Placement and Scheduling of Sensors*), that jointly optimizes the sensor placement and the sensor schedule. We prove that our algorithm provides a constant factor approximation to the optimal solution of this NP-hard optimization problem.

Most existing approaches to sensor placement and scheduling associate a fixed sensing region with every sensor, and then attempt to maximize the number of regions covered in every group of sensors (*c.f.*, Abrams et al. (2004); Deshpande et al. ('08); Hochbaum and Maas (1985)). In complex applications such as traffic or environmental monitoring however, the goal of sensor placement is a prediction problem, where one intends to predict the sensed phenomenon at the locations where no sensors are placed. Our algorithm applies to such settings where the sensing quality of a set of sensors is measured, e.g., in the improvement of prediction accuracy (more formally, our algorithm applies whenever the sensing quality function satisfies *submodularity*, an intuitive diminishing returns property).

In contrast to most existing algorithms that optimize scheduling for average case performance, our approach furthermore provides a schedule that performs uniformly well over time, hence leading to a

well-balanced performance of the sensor network. For security-critical applications such as outbreak detection, such balanced performance is a crucial requirement not met by existing algorithms. In fact, our experimental results show that average-case optimal solutions can lead to arbitrarily unbalanced performance, but optimizing for balanced performance (using eSPASS) typically leads to good average-case performance.

Deploying a large number of scheduled sensors has the additional benefit that it allows trading off power and accuracy. The deployed network might have several modes of operation: a scheduled mode of operation, where only a small fraction of sensors is turned on, and a "high density" mode where all (or a larger fraction of) sensors are activated. For example, in traffic monitoring, once a traffic congestion is detected (during scheduled mode), the high density mode could be used to accurately identify the boundary of the congestion. We show how our algorithm can be extended to support such a power-accuracy tradeoff.

We present extensive empirical studies on several case studies, illustrating the versatility of our algorithm. These case studies include sensing tasks such as traffic and environmental monitoring and placing sensors for outbreak detection and selecting informative weblogs to read on the Internet. Our results show that simultaneously placing and scheduling results in drastically improved performance compared to the setting where optimization over the placement and the scheduling are performed separately.

In summary, our main contributions are:

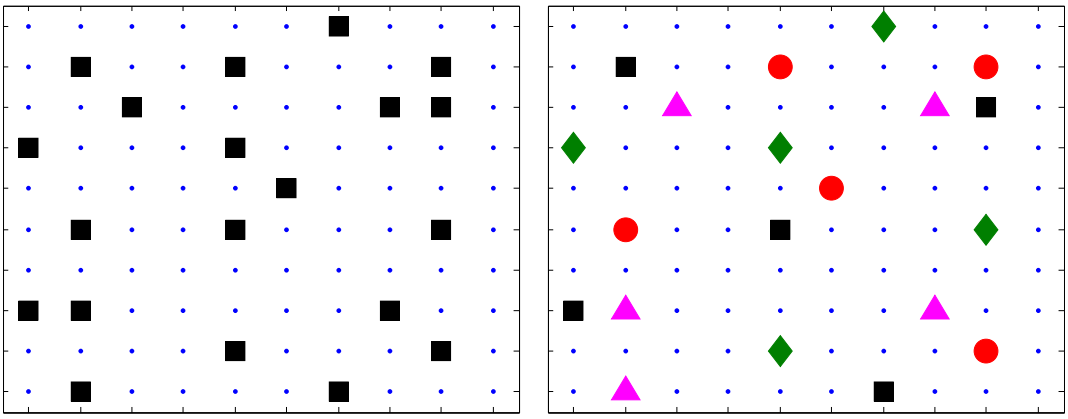
- We study the problem of simultaneously placing and scheduling sensors as a novel optimization problem.
- We develop eSPASS, an efficient approximation algorithm for this problem, that applies to a variety of realistic sensing quality functions (such as area coverage, variance reduction, outbreak detection, etc.). Our algorithm is guaranteed to provide a near-optimal solution, that obtains at least a constant fraction of the optimal sensing quality. eSPASS furthermore allows to trade off power consumption and accuracy.
- We perform several extensive case studies on real sensing problems in traffic and environmental monitoring as well as outbreak detection, demonstrating the effectiveness of our approach.

2 Problem Statement

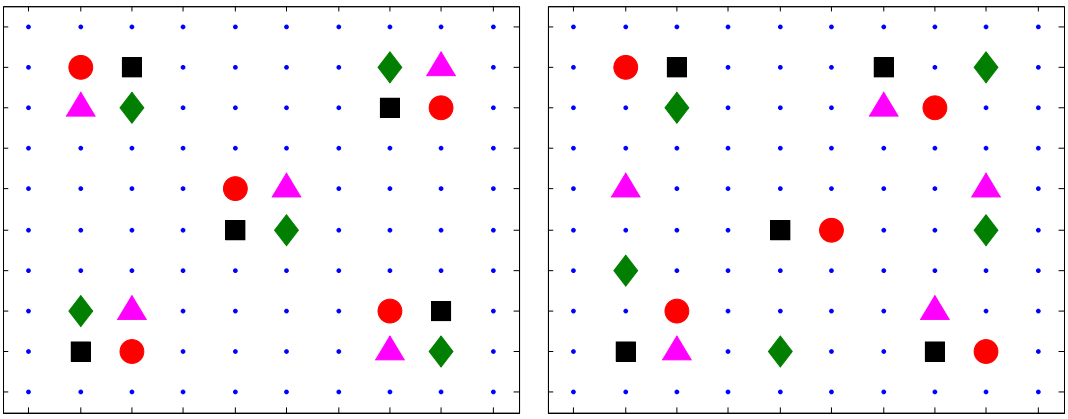
We will first separately introduce the sensor placement and scheduling problems, and then formalize the problem of simultaneously placing and scheduling sensors.

2.1 Sensor Placement

In *sensor placement*, we are given a finite set $\mathcal{V}$ of possible locations where sensors can be placed. Our goal is to select a small subset $\mathcal{A} \subseteq \mathcal{V}$ of locations to place sensors at, that maximizes a sensing quality function $F(\mathcal{A})$. There are several different notions of sensing quality that we might want to optimize, each depending on the particular sensing task. For example, we can associate sensing regions with every sensor, and $F(\mathcal{A})$ can measure the total area covered when placing sensors at locations $\mathcal{A}$. In complex applications such as the traffic monitoring problem, we are interested in optimizing the prediction accuracy when obtaining measurements from locations $\mathcal{A}$. In this setting, we can model the state of the world (e.g., the traffic condition at different locations) using a collection of random variables $\mathcal{X}_\mathcal{V}$, one variable $\mathcal{X}_s$ for each location $s \in \mathcal{V}$. We can then use a probabilistic model (such as a Gaussian Process which is frequently used in geostatistics, *c.f.*, Cressie (1991)) that models a joint probability distribution $P(\mathcal{X}_\mathcal{V})$ over the possible locations. Upon acquiring measurements $\mathcal{X}_\mathcal{A} = \mathbf{x}_\mathcal{A}$ at a subset of locations $\mathcal{A}$, we can then predict the phenomenon at the



(a) *Optimized sensor placement* (b) *Optimized schedule for placement (a)*



(c) *Simultaneous placement and schedule* (d) *Multicriterion solution*

Figure 2: In the stage-wise approach, sensors are first deployed (a), and the deployed sensors are then scheduled (b, sensors assigned to the same time slot are drawn using the same color and marker). In the simultaneous approach, we jointly optimize over placement and schedule (c). (d) Multicriterion solution to Problem (4) ($\lambda = .25$) that performs well both in scheduled and high-density mode.

unobserved locations using the conditional distribution $P(\mathcal{X}_{\mathcal{V}\setminus\mathcal{A}} \mid \mathcal{X}_{\mathcal{A}} = \mathbf{x}_{\mathcal{A}})$. We can then use the expected mean squared error,

$$\mathrm{Var}(\mathcal{X}_{\mathcal{V}} \mid \mathcal{X}_{\mathcal{A}} = \mathbf{x}_{\mathcal{A}}) = \frac{1}{|\mathcal{V}|} \sum_{s \in \mathcal{V}} \mathbb{E}\left[(\mathcal{X}_s - \mathbb{E}[\mathcal{X}_s \mid \mathbf{x}_{\mathcal{A}}])^2 \mid \mathbf{x}_{\mathcal{A}}\right]$$

to quantify the uncertainty in this prediction. Since we do not know the values $\mathbf{x}_{\mathcal{A}}$ before placing the sensors, a natural choice of the sensing quality function $F(\mathcal{A})$ is to measure the *expected reduction in variance* at the unobserved locations,

$$F(\mathcal{A}) = \mathrm{Var}(\mathcal{X}_{\mathcal{V}}) - \int P(\mathbf{x}_{\mathcal{A}}) \, \mathrm{Var}(\mathcal{X}_{\mathcal{V}} \mid \mathcal{X}_{\mathcal{A}} = \mathbf{x}_{\mathcal{A}}) d\mathbf{x}_{\mathcal{A}}.$$

This sensing quality function has been found useful for sensor selection (*c.f.*, Deshpande et al. (2004); Krause et al. (2008a)) and experimental design (*c.f.*, Chaloner and Verdinelli (1995)).

It can be shown that both the area covered and the variance reduction objective, as well as many other notions of sensing quality, satisfy the following intuitive diminishing returns property (Das

and Kempe, 2008; Krause et al., 2007)[1]: Adding a sensor helps more if we have placed few sensors so far, and less if we already have placed lots of sensors. This intuition can be formalized using the combinatorial concept of *submodularity*: A set function F is called submodular, if for all $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$ and $s \in \mathcal{V} \setminus \mathcal{B}$

$$F(\mathcal{A} \cup \{s\}) - F(\mathcal{A}) \geq F(\mathcal{B} \cup \{s\}) - F(\mathcal{B}),$$

i.e., adding s to a small set $\mathcal{A}$ helps more than adding s to the superset $\mathcal{B}$. In addition, these sensing quality functions are *monotonic*: For all $\mathcal{A} \subseteq \mathcal{B}$ it holds that $F(\mathcal{A}) \leq F(\mathcal{B})$, i.e., adding more sensors can only improve the sensing quality.

Based on this notion of a monotonic, submodular sensing quality function, the sensor placement problem then is

$$\max_{\mathcal{A}} F(\mathcal{A}) \text{ such that } |\mathcal{A}| \leq m,$$

i.e., we want to find a set $\mathcal{A}$ of at most m locations to place sensors maximizing the sensing quality F.

2.2 Sensor Scheduling

In *sensor scheduling*, we are given a sensor placement (i.e., locations $\mathcal{A}$), and our goal is to assign each sensor $s \in \mathcal{A}$ one of k time slots. This assignment partitions the set $\mathcal{A}$ into disjoint sets $\mathcal{A}_1, \ldots, \mathcal{A}_k$, where $\mathcal{A}_t \subseteq \mathcal{A}$ is the subset of sensors that have been assigned slot t. A round-robin schedule can then be applied that cycles through the time slots, and activates sensors $\mathcal{A}_t$ at time t. Since each sensor is active at only one out of k time slots, this procedure effectively increases the lifetime of the network by a factor of k. How can we quantify the value of a schedule $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_k)$? For each group $\mathcal{A}_t$, we can compute the sensing quality $F(\mathcal{A}_t)$[2]. One possibility would then be to optimize for the *average* performance over time,

$$\max_{\mathcal{A}_1,\ldots,\mathcal{A}_k} \frac{1}{k} \sum_{t=1}^{k} F(\mathcal{A}_t).$$

However, as we show in our experiments, if we optimize for the average case performance, it can happen that a few of the time slots are very poorly covered, i.e., there is a time t such that $F(\mathcal{A}_t)$ is very low. For security-critical applications, this can be problematic. Instead, we can also optimize for a *balanced* schedule,

$$\max_{\mathcal{A}_1,\ldots,\mathcal{A}_k} \min_{t} F(\mathcal{A}_t),$$

that performs uniformly well over time.

Note that the above formulation of the scheduling problem allows to handle settings where each sensor can be active at $r \geq 1$ timesteps. In this setting, we simply define a new ground set $\mathcal{A}' = \mathcal{A} \times \{1, \ldots, r\}$ where the pair $(s, i) \in \mathcal{A}'$ refers to the i-th activation of sensor s. The sensing quality function is modified as $F'(\mathcal{A}'_j) = F(\{s : \exists_i (s, i) \in \mathcal{A}'_j\})$.

2.3 Simultaneous placement and scheduling

Both sensor placement and sensor scheduling have been studied separately from each other in the past. One approach towards placement and scheduling would be to first use an algorithm (such as the algorithm proposed by Krause et al. (2007)) to find a sensor placement $\mathcal{A}$, and then use a separate algorithm (such as the mixed integer approach of Koushanfary et al. (2006)) to find a

[1] Variance reduction has been shown to be submodular for Gaussian distributions under certain assumptions about the covariance by Das and Kempe (2008).

[2] Note that we assume the same sensing quality function F for each time step. This assumption has been made in the past (*c.f.*, Koushanfary et al. (2006); Abrams et al. (2004)), and is reasonable for many monitoring tasks.

schedule $\mathcal{A}_1, \ldots, \mathcal{A}_k$. We call this approach a *stage-wise* approach, and illustrate it in Figures 2(a) and 2(b).

Instead of separating placement and scheduling, we can *simultaneously* optimize for the placement and the schedule. Suppose we have resources to purchase m sensors, and we would like to extend the network lifetime by a factor of k. Our goal would then be to find k disjoint sets $\mathcal{A}_1, \ldots, \mathcal{A}_k \subseteq \mathcal{V}$, such that together these sets contain at most m locations, i.e., $|\bigcup_t \mathcal{A}_t| \leq m$. We call this problem the SPASS problem, for *simultaneous placement and scheduling of sensors*. Again, we can consider the average-case performance,

$$\max_{\mathcal{A}_1 \ldots \mathcal{A}_k} \frac{1}{k} \sum_{t=1}^{k} F(\mathcal{A}_t) \text{ s.t. } \mathcal{A}_i \cap \mathcal{A}_j = \emptyset \text{ if } i \neq j \text{ and } |\bigcup_t \mathcal{A}_t| \leq m \tag{1}$$

and the balanced objective:

$$\max_{\mathcal{A}_1 \ldots \mathcal{A}_k} \min_t F(\mathcal{A}_t) \text{ s.t. } \mathcal{A}_i \cap \mathcal{A}_j = \emptyset \text{ if } i \neq j \text{ and } |\bigcup_t \mathcal{A}_t| \leq m. \tag{2}$$

By performing this simultaneous optimization, we can obtain very different solutions, as illustrated in Figure 2(c). In Section 6, we will show that this simultaneous approach can lead to drastically improved performance as compared to the traditional, stage-wise approach. In this paper, we present eSPASS, an efficient approximation algorithm with strong theoretical guarantees for this problem.

The placement and schedule in Figure 2(c) has the property that the sensors selected at each time step share very similar locations, and hence perform roughly equally well. However, if activated all at the same time, the "high-density" performance $F(\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_k)$ is much lower than that of the placement in Figure 2(a). We also develop an algorithm, mcSPASS, that leads to placements which perform well both in scheduled and in high-density mode. Figure 2(d) presents the solution obtained for the mcSPASS algorithm.

Note that instead of fixing the number of time slots, we could also specify a desired accuracy constraint Q and then ask for the maximum lifetime solution, i.e., the largest number k of time slots such that a solution with minimum (or average) sensing quality Q is obtained. Clearly, an algorithm that solves Problem (2) (or Problem (1)) could be used to solve this alternative problem, by simply binary searching over possible values for k[3].

Further note that it is possible to allow each sensor to be active at $r \geq 1$ timesteps by using the modification described in Section 2.2.

3 A naive greedy algorithm

We will first study the problem of optimizing the average performance over time, i.e., Problem (1), for a fixed monotonic submodular sensing quality function F. Considering the fact that simultaneously placing and scheduling is a strict generalization of sensor placement, which itself is NP-hard (*c.f.*, Krause et al. (2007)), we cannot expect to efficiently find the optimal solution to Problem (1) in general.

Instead, we will use the following intuitive greedy algorithm that we call GAPS for *Greedy Average-case Placement and Scheduling*. At every round, GAPS picks a time slot t and location s which increases the total sensing quality the most, until m location/time-slot pairs have been picked. It is formalized as Algorithm 1.

[3]However, in case an approximate algorithm is used such as the eSPASS algorithm developed in this paper, its guarantees are not necessarily preserved.

```
   Algorithm GAPS (F, V, k, m)
   A_t ← ∅ for all t;
   for i = 1 to m do
      foreach s ∈ V \ (A_1 ∪ ··· ∪ A_k), 1 ≤ t ≤ k do
1        δ_{t,s} ← F(A_t ∪ {s}) − F(A_t);
      end
      (t*, s*) ← argmax_{t,s} δ_{t,s};
      A_{t*} ← A_{t*} ∪ {s*};
   end
```

Algorithm 1: The greedy average-case placement and scheduling (GAPS) algorithm.

3.1 Theoretical guarantee

Perhaps surprisingly, we can show that this simple algorithm provides near-optimal solutions for Problem (1). In fact, it generalizes the distributed Set-k Cover algorithm proposed by Abrams et al. (2004) to arbitrary submodular sensing quality functions F, and to the setting where at most m sensors can be selected in total.

Theorem 3.1. *For any monotonic, and submodular function F,* GAPS *returns a solution $\mathcal{A}_1, \ldots, \mathcal{A}_k$ s.t.*

$$\frac{1}{k}\sum_t F(\mathcal{A}_t) \geq \frac{1}{2}\max_{\mathcal{A}'}\frac{1}{k}\sum_t F(\mathcal{A}'_t).$$

GAPS *requires at most $\mathcal{O}(kmn)$ evaluations of F.*

The proofs of Lemma 4.2 and all other results are given in the Appendix. The key observation is that Problem (1) is an instance of maximizing a submodular function subject to a *matroid* constraint (*c.f.*, the Appendix for details). A fundamental result by Fisher et al. (1978) then proves that the greedy algorithm returns a solution that obtains at least one half of the optimal average-case score. Matroids for sensor scheduling have been considered before by Williams et al. (2007).

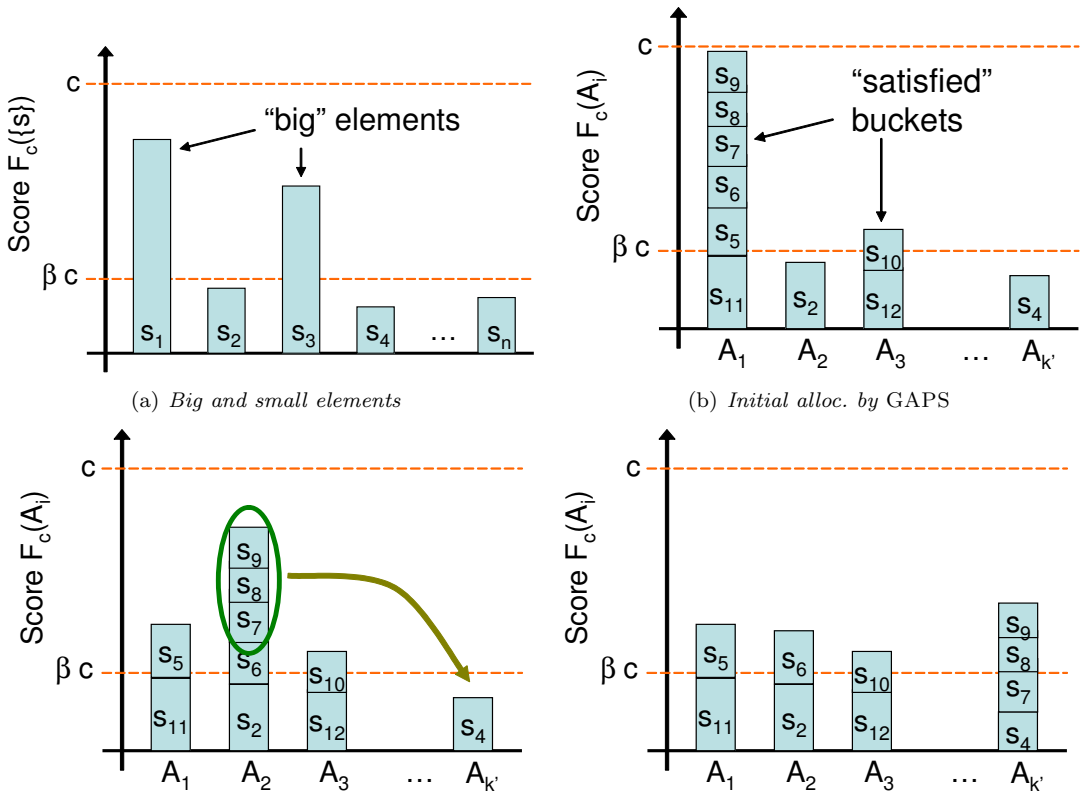
3.2 Greedy can lead to unbalanced solutions

If a sensor placement and schedule is sought that performs well "on-average" over time, GAPS performs well. However, even though the average performance over time, $\frac{1}{k}\sum_t F(\mathcal{A}_t)$, is high, the performance at some individual timesteps t' can be very poor, and hence the schedule can be *unfair*. In security-critical applications, where high performance is required at all times, this behavior can be problematic. In such settings, we might be interested in optimizing the *balanced* performance over time, $\min_t F(\mathcal{A}_t)$. This optimization task was raised as an open problem by (Abrams et al., 2004).

A first idea would be to try to modify the GAPS algorithm to directly optimize this balanced performance, i.e., replace Line 1 in Algorithm 1 by

$$\delta_{t,s} \leftarrow \min_j F(\mathcal{A}_j^{+(t,s)}) - \min_j F(\mathcal{A}_j),$$

where $\mathcal{A}^{+(t,s)}$ is the solution obtained by adding location s to time slot $\mathcal{A}_t$ in solution $\mathcal{A} = \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_k$. We call this modified algorithm the GBPS algorithm (for Greedy Balanced Placement and Scheduling). Unfortunately, both GAPS and GBPS can perform arbitrarily badly. Consider a simple scenario with three locations, $\mathcal{V} = \{a, b, c\}$, and the monotonic submodular function $F(\mathcal{A}) = |\mathcal{A}|$. We want to partition $\mathcal{V}$ into three timesteps, i.e., $k = 3$ and $m = 3$. Here, the optimal solution would be to pick $\mathcal{A}_1^* = \{a\}$, $\mathcal{A}_2^* = \{b\}$ and $\mathcal{A}_3^* = \{c\}$. However, both GAPS and GBPS would (ties broken unfavorably) pick $\mathcal{A}_1 = \{a, b, c\}$ and $\mathcal{A}_2 = \mathcal{A}_3 = \emptyset$, obtaining a minimum score of 0.



(a) *Big and small elements* (b) *Initial alloc. by* GAPS

(c) *Reallocation move* (d) *Reallocated elements*

Figure 3: Illustration of our eSPASS algorithm. The algorithm first "guesses" (binary searches for) the optimal value c. (a) Then, big elements s where $F(\{s\}) \geq \beta c$ are allocated to separate buckets. (b) Next, the remaining small elements are allocated to empty buckets using the GAPS algorithm. (c,d) Finally, elements are reallocated until all buckets are satisfied.

Unfortunately, this poor performance is not just a theoretical example – in Section 6 we demonstrate it empirically on real sensing tasks.

4 The eSPASS algorithm

In the following, we will develop an efficient algorithm, eSPASS (for *efficient Simultaneous Placement and Scheduling of Sensors*), that, as we will show in Section 4.2, is guaranteed to provide a near-optimal solution to the Problem (2). To the best of our knowledge, our algorithm is the first algorithm with theoretical guarantees for this general problem, hence partly resolving the open problem described by Abrams et al. (2004).

4.1 Algorithm overview

We start with an outline of our algorithm, and then proceed to discuss each step more formally.

Our high-level goal will be to reduce the problem of optimizing the balanced objective into a sequence of modified optimization problems involving an average-case objective, which we can approximately solve using GAPS. This idea is based on the following intuition: Consider a *truncated*

objective function $F_c(\mathcal{A}) = \min\{F(\mathcal{A}), c\}$. The key observation[4] is that, for *any* constant c, it holds that

$$\min_t F(\mathcal{A}_t) \geq c \Leftrightarrow \frac{1}{k}\sum_{t=1}^{k} F_c(\mathcal{A}_t) = c,$$

i.e., the minimum score is greater than or equal to c if and only if the average truncated score is c.

Now suppose someone tells us the value c^* attained by an optimal solution, i.e., $\max_{\mathcal{A}} \min_t F(\mathcal{A}_t) = c^*$. By the above observation this problem is equivalent to solving

$$\max_{\mathcal{A}} \frac{1}{k}\sum_{t=1}^{k} F_{c^*}(\mathcal{A}_t). \tag{3}$$

It can be shown (*c.f.*, Fujito (2000)) that for $c \geq 0$, the truncated objective function F_c remains monotonic and submodular. Hence, Problem (3) is an instance of the *average-case* Problem (1). Now we face the challenge that we do *not* generally know the optimal value c^*. We could use a simple binary search procedure to find this optimal value. Hence, if we could *optimally* solve the monotonic submodular *average-case* Problem (1), we would obtain an optimal solution to the *balanced* Problem (2).

Unfortunately, as shown in Section 3.1, solving the average-case problem is NP-hard, and, using GAPS, we can only solve it *approximately*, obtaining a solution that achieves at least half of the optimal value. In the following, we will show how we can turn this approximate solution for the average-case problem into a near-optimal solution for the balanced problem.

Our algorithm will maintain one "bucket" $\mathcal{A}_t \subseteq \mathcal{V}$ for each time slot t. Since our goal is to develop an approximation algorithm achieving at least a fraction $\beta > 0$ of the optimal sensing quality, we need to allocate m elements $s \in \mathcal{V}$ to the k buckets such that $F(\mathcal{A}_t) \geq \beta c^*$ for all buckets $\mathcal{A}_t$. Hereby, β is a constant that we will specify later. We call a bucket "satisfied" if $F(\mathcal{A}_t) \geq \beta c^*$, "unsatisfied" otherwise. Here is an outline of our eSPASS algorithm, Figure 3 presents an illustration.

1. "Guess" the optimal value c.
2. Call an element $s \in \mathcal{V}$ "big" if $F_c(\{s\}) \geq \beta c$ and "small" otherwise. Put each big element into a separate bucket (*c.f.*, Figure 3(a)). From now on, we ignore those satisfied buckets, and focus on the unsatisfied buckets.
3. Run GAPS to optimize F_c and allocate the small elements to the unsatisfied buckets (*c.f.*, Figure 3(b)).
4. Pick a "satisfied" bucket $\mathcal{A}_t$ that contains sufficiently many elements, and reallocate enough elements to an "unsatisfied" bucket to make it satisfied (*c.f.*, Figures 3(c) and 3(d)). Repeat step 3 until no more buckets are unsatisfied or no more reallocation is possible. We will show that this reallocation will always terminate.
5. If all buckets are satisfied, return to step 1 with a more optimistic (higher) "guess" for c. If at least one bucket remains unsatisfied, return to step 1 with a more pessimistic (lower) guess for c.

eSPASS terminates with a value for c such that *all* buckets t have been assigned elements $\mathcal{A}_t$ such that $F(\mathcal{A}_t) \geq \beta c$. It guarantees that upon termination, c is an upper bound on the value of the optimal solution, hence providing a β approximation guarantee. In Section 4.2 we will show that $\beta = \frac{1}{6}$ suffices. In summary, we have the following guarantee about eSPASS:

[4]Krause et al. ('08) used this observation to develop an algorithm for robust optimization of submodular functions.

Theorem 4.1. *For any monotonic, submodular function F and constant $\varepsilon > 0$,* eSPASS*, using* GAPS *as subroutine, returns a solution $\mathcal{A}_1, \ldots, \mathcal{A}_k$ such that*

$$\min_t F(\mathcal{A}_t) \geq \frac{1}{6} \max_{\mathcal{A}'} \min_t F(\mathcal{A}'_t) - \varepsilon.$$

eSPASS *requires at most $\mathcal{O}\left((1 + \log_2 F(\mathcal{V})/\varepsilon)kmn\right)$ evaluations of F.*

Hereby, ε is an tolerance parameter that can be made arbitrarily small. The number of iterations increases only logarithmically in $1/\varepsilon$.

```
   Algorithm eSPASS (F, V, k, m, ε)
   c_min ← 0; c_max ← F(V); β ← 1/6;
   while c_max − c_min ≥ ε do
       c ← (c_max + c_min)/2;
1      B ← {s ∈ V : F_c({s}) ≥ βc};
       k' ← k;
2      foreach s ∈ B do
           A_{k'} ← {s}; k' ← k' − 1;
           if k' = 0 then c_min ← c;
               A_best ← (A_1, ..., A_k);
               continue with while loop;
       end
       V' ← V \ B; m' ← m − |B|;
3      A_{1:k'} ← GAPS(F_c, V', k', m');
4      if Σ_t F(A_t) < k'c/2 then c_max ← c; continue;
       else
5          while ∃i, j ≤ k': F_c(A_j) ≤ βc, F_c(A_i) ≥ 3βc do
               foreach s ∈ A_i do
                   A_j ← A_j ∪ {s}; A_i ← A_i \ {s};
                   if F_c(A_j) ≥ βc then break;
               end
           end
           c_min ← c; A_best ← (A_1, ..., A_k);
       end
   end
```

Algorithm 2: The eSPASS algorithm for simultaneously placing and scheduling sensors.

4.2 Algorithm details

We will now analyze each of the steps of eSPASS in detail. The pseudocode is given in Algorithm 2.

Removing big elements. The main challenge when applying the GAPS algorithm to the truncated Problem (3) is exemplified by the following pathological example. Suppose the optimal value is c. GAPS, when applied to the truncated function F_c, could pick $k/2$ elements $s_1, \ldots, s_{k/2}$, with $F(\{s_i\}) = c$ each. While this solution obtains an average-case score of $c/2$ (one half of optimal as guaranteed by Theorem 3.1), there is no possibility to reallocate these $k/2$ elements into k buckets, and hence some buckets will remain empty, giving a balanced score of 0.

To avoid this pathological case, we would like to eliminate such elements $s \in \mathcal{V}$ with high individual scores $F(\{s\})$, to make sure that we can rearrange the solution of GAPS to obtain high balanced score. Hence, we distinguish two kinds of elements: Big elements s with $F(\{s\}) \geq \beta c$, and small elements s with $F(\{s\}) < \beta c$. If we intend to obtain a β approximation to the optimal

score c, we realize that big elements have high enough value to each satisfy an individual bucket. Let $\mathcal{B}$ be the set of big elements (this set is determined in Line 1). If $|\mathcal{B}| \geq k$, we already have a β-approximate solution: Just put one big element in each bucket. If $|\mathcal{B}| < k$, put each element in $\mathcal{B}$ in a separate bucket $\mathcal{A}_1, \ldots, \mathcal{A}_{|\mathcal{B}|}$ (*c.f.*, Line 2). We can now set these satisfied buckets aside, and look at the reduced problem instance with elements $\mathcal{V}' = \mathcal{V} \setminus \mathcal{B}$, $m' = m - |\mathcal{B}|$ and $k' = k - |\mathcal{B}|$. Our first lemma shows that if the original problem instance $(F, \mathcal{V}, k, m)$ has optimal value c, the reduced problem instance $(F, \mathcal{V}', k', m')$ still has optimal value c.

Lemma 4.2. *The optimal value on the new problem instance $(F, \mathcal{V}', k', m')$ is still c.*

Hence, without loss of generality, we can now assume that for all $s \in \mathcal{V}$, $F(\{s\}) \leq \beta c$.

Solving the average-case problem. In the next step of the algorithm, we run an α-approximate algorithm (such as GAPS where $\alpha = \frac{1}{2}$), using the truncated objective F_c, on the reduced problem instance containing only small elements (*c.f.*, Line 3). This application results in an allocation $\mathcal{A}_1, \ldots, \mathcal{A}_{k'}$ of elements into buckets. If $\sum_t F_c(\mathcal{A}_t) < \alpha c k'$, then we know that c is an upper bound to the optimal solution, and it is safe to set c_{max} to c (*c.f.*, Line 4) and continue with the binary search. Otherwise, we have a solution where $\sum_t F_c(\mathcal{A}_t) \geq \alpha c k'$. However, as argued in Section 3.2, this α-approximate solution could still have balanced score 0, if all the elements are allocated to only the first $\alpha k'$ buckets. Hence, we need to reallocate elements from satisfied into unsatisfied buckets to obtain a balanced solution.

Reallocation. We will transfer elements from satisfied buckets to unsatisfied buckets, until all buckets are satisfied. Let us define a "reallocation move" as follows (*c.f.*, Line 5). Pick a bucket $\mathcal{A}_i = \{a_1, \ldots, a_l\}$ for which $F_c(\mathcal{A}_i) \geq 3\beta c$ (we will guarantee that such a bucket always exists), and a bucket $\mathcal{A}_j$ that is not satisfied, i.e., $F_c(\mathcal{A}_j) < \beta c$. Choose ℓ such that $F_c(\{a_1, \ldots, a_{\ell-1}\}) < \beta c$ and $F_c(\{a_1, \ldots, a_\ell\}) \geq \beta c$. Let $\Delta = \{a_1, \ldots, a_\ell\}$. Note that Δ is not empty since each a_i is small (i.e., $F_c(\{a_i\}) < \beta c$). We reallocate the elements Δ by removing Δ from $\mathcal{A}_i$ and adding Δ to $\mathcal{A}_j$.

Lemma 4.3. *It holds that*

$$F_c(\mathcal{A}_j \cup \Delta) \geq \beta c, \text{ and } F_c(\mathcal{A}_i \setminus \Delta) \geq F_c(\mathcal{A}_i) - 2\beta c.$$

Hence, removing elements Δ does not decrease the value of $\mathcal{A}_i$ by more than $2\beta c$, and thus $\mathcal{A}_i$ remains satisfied. On the other hand, the previously unsatisfied bucket $\mathcal{A}_j$ becomes satisfied by adding the elements Δ. We want to make sure that we can always execute our reallocation move, until all buckets are satisfied. The following result shows that if we choose $\beta = \frac{\alpha}{3}$, this will always be the case:

Lemma 4.4. *If we set $\beta = \frac{\alpha}{3}$, then, after at most k reallocation moves, all buckets will be satisfied, i.e., $F_c(\mathcal{A}_i) \geq \beta c$ for all i.*

Binary search. Since the optimal value c is generally not known, we have to search for it. This is done using a simple binary search strategy, starting with the interval $[0, F(\mathcal{V})]$ which is guaranteed the optimal value due to monotonicity. At every step, we test the center c of the current interval. If all buckets can be filled to βc, then the truncation threshold c can be increased. If the algorithm for maximizing the average-case score (such as GAPS) does not return a solution of value at least αc, then that implies that the optimal value has to be less than c, and the truncation threshold is decreased (*c.f.*, Line 4). If F takes only integral values, then after at most $\lceil \log_2 F(\mathcal{V}) \rceil + 1$ iterations, the binary search terminates.

4.3 Improving the bounds

The bound in Theorem 4.1 is "offline" – we can state it independently of the specified problem instance. While guaranteeing that the obtained solutions cannot be arbitrarily bad, the constant

factor 6 bound for eSPASS is typically rather weak, and we can show that our obtained solutions are typically much closer to the optimal value.

We can do this by computing the following data-dependent bounds on the optimal value. Let $\mathcal{A}' = (\mathcal{A}'_1, \ldots, \mathcal{A}'_k)$ be a candidate solution to Problem (2) (e.g., obtained using the eSPASS algorithm or any other algorithm). For every $1 \leq \ell \leq k$ and $s \in \mathcal{V}$ let $\delta_{\ell,s} = F(\mathcal{A}'_\ell \cup \{s\}) - F(\mathcal{A}'_\ell)$ be the increment in function value when adding sensor s to time slot ℓ.

Theorem 4.5. *The optimal value $c^* = \max_{\mathcal{A}} \min_t F(\mathcal{A}_t)$ is bounded by the solution c to the following linear program:*

$$\max_{\lambda_{i,s},c} c \text{ s.t.}$$
$$c \leq F(\mathcal{A}_i) + \sum_s \lambda_{i,s}\delta_{i,s} \text{ for all } i$$
$$\sum_i \lambda_{i,s} \leq 1 \text{ for all } s \text{ and } \sum_{i,s} \lambda_{i,s} \leq m$$

Theorem 4.5 states that for any given instance of the SPASS problem, and for a candidate solution $\mathcal{A}$ obtained by using *any* algorithm (not necessarily using eSPASS), we can solve a linear program to efficiently get an upper bound on the optimal solution. In Section 6 we will show that these bounds prove that our solutions obtained using eSPASS are often much closer to the optimal solution than guaranteed by the bound of Theorem 4.1.

5 Trading off Power and Accuracy

As argued in Section 1, deploying a larger number of sensors and scheduling them has the advantage over deploying a small number of sensors with large batteries that a high density mode can be supported. In contrast to scheduled mode, where the sensors are activated according to the schedule, in high-density mode all sensors are active, to provide higher resolution sensor data (e.g., to localize the boundary of a traffic congestion in our running example). For a fixed solution $\mathcal{A}_1, \ldots, \mathcal{A}_k$ to the SPASS problem, the (balanced) scheduled-mode sensing quality is $\min_i F(\mathcal{A}_i)$, whereas the high-density sensing quality is $F(\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_k)$. Note that optimizing for the scheduled sensing quality does not necessarily lead to good high-density sensing quality. Hence, if both modes of operation should be supported, then we should simultaneously optimize for both performance measures. One such approach to this multicriterion optimization problem is to define the scalarized objective

$$\widehat{F}_\lambda(\mathcal{A}_1, \ldots, \mathcal{A}_k) = \lambda \min_i F(\mathcal{A}_i) + (1-\lambda)F(\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_k),$$

and then solve the problem

$$\max_{\mathcal{A}_1 \ldots \mathcal{A}_k} \widehat{F}_\lambda(\mathcal{A}_1, \ldots, \mathcal{A}_k) \text{ s.t. } \mathcal{A}_i \cap \mathcal{A}_j = \emptyset \text{ if } i \neq j, \, |\bigcup_t \mathcal{A}_t| \leq m. \tag{4}$$

Note that if $\lambda = 1$, we recover the SPASS problem. Furthermore, as $\lambda \to 0$, the high-density sensing quality $F(\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_k)$ dominates, and the chosen solution will converge to the stage-wise approach, where first the set $\mathcal{A}$ of all sensors is optimized, and then this placement is partitioned into $\mathcal{A} = \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_k$. Hence, by varying λ between 1 and 0, we can interpolate between the simultaneous and the stage-wise placement and scheduling.

We modify eSPASS to approximately solve Problem (4), and call the modified algorithm mcSPASS (for *multicriterion Simultaneous Placement and Scheduling of Sensors*). The basic strategy is still a binary search procedure. However, instead of simply picking all available big elements (as done by eSPASS), mcSPASS will also guess (search for) the number ℓ of big elements used in the optimal solution. It will pick these big elements in a greedy fashion, resulting in a set $\mathcal{A}_{big} \subseteq \mathcal{V}$. For

a fixed guess of c and ℓ, mcSPASS will again use GAPS as a subroutine. However, the objective function used by GAPS will be modified to account for the high-density performance:

$$G(\mathcal{A}_1, \ldots, \mathcal{A}_{k'}) = \lambda \sum_i F_c(\mathcal{A}_i) + (1-\lambda) F(\mathcal{A} \cup \mathcal{A}_{big}),$$

where $\mathcal{A} = \mathcal{A}_1 \cup \ldots \mathcal{A}_{k'}$, and $k' = k - \ell$. This modified objective function combines a component (weighted by λ) that measures the scheduled performance, as well as a component (weighted by $1-\lambda$) that measures the improvement in high-density performance, taking into account the set $\mathcal{A}_{big}$ of big elements that have already been selected. The reallocation procedure remains the same as in eSPASS. The remaining details of our mcSPASS approach are presented in the proof to the following theorem, which can be found in the Appendix.

Theorem 5.1. *For any monotonic, submodular function F and constants $\varepsilon > 0$ and $0 \leq \lambda \leq 1$,* mcSPASS *will efficiently find a solution $\mathcal{A}_1, \ldots, \mathcal{A}_k$ such that*

$$\widehat{F}_\lambda(\mathcal{A}_1, \ldots, \mathcal{A}_k) \geq \frac{1}{8} \max_{\mathcal{A}'} \widehat{F}_\lambda(\mathcal{A}'_1, \ldots, \mathcal{A}'_k) - \varepsilon.$$

In Section 6 we will see that we can use this extension to obtain placements and schedules that perform well both in scheduled and high-density mode.

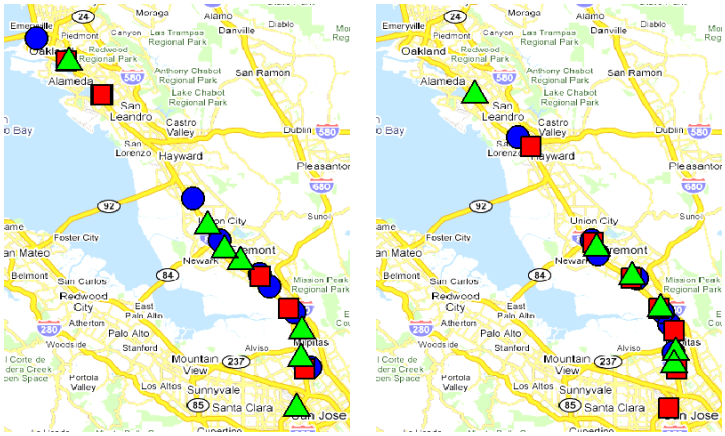
6 Experiments

In our experiments, we analyze several case studies on different data sets. Each case study presents a different justification of the SPASS optimization problem.

6.1 Case study I: Highway monitoring

The California highways are currently monitored by over 10,000 traffic sensors based on older technologies. As these loops fail, they are being replaced by novel wireless sensor networks technologies, and it is an important problem to identify economic deployment strategies. PeMS (Berkeley) is a website and project that integrates, cleanses and tracks real time traffic information for the whole state, computing key performance indicators. The sensors typically report speed, flow and vehicle counts every 30 seconds, and PeMS aggregates the data further into 5 minute blocks. For this case study, we use data from highway I-880 South, which extends for 35 miles in northern California (Figure 4) and has between 3 and 5 lanes. This highway experiences heavy traffic, and accurate measurements are essential for proper resource management. Measurement variation is mainly due to congestion and events such as accidents and road closures. There are 88 measurement sites along the highway, on average every 2 miles, which comprise 357 sensors covering all lanes. We use speed information from lanes, for all days of the week in a single month, excluding weekends and holidays for the period from 6AM to 11AM, which is the time when the highway is congested. This is the most difficult time for making predictions, as when there is no congestion, even a free flow speed prediction of 60 mph is accurate.

The number and locations of sensors are limited by costs and physical deployment constraints. Typically at each location, it is only possible to place one sensor at each lane. Furthermore, lane closures for sensor installations are very costly. Given these constraints, California requires that sensor technologies have a target lifetime of 10 years. This implies that most wireless sensor solutions require intelligent scheduling in order to extend the lifetime by four times, since most sensor network solutions batteries are expected to last 2 to 3 years. Including more batteries in a single sensor is not viable, as sensors have physical constraints to avoid disrupting the existing pavement structure and keep installation costs at a minimum.



(a) *[T] stage-wise* (b) *[T] Simultaneous*

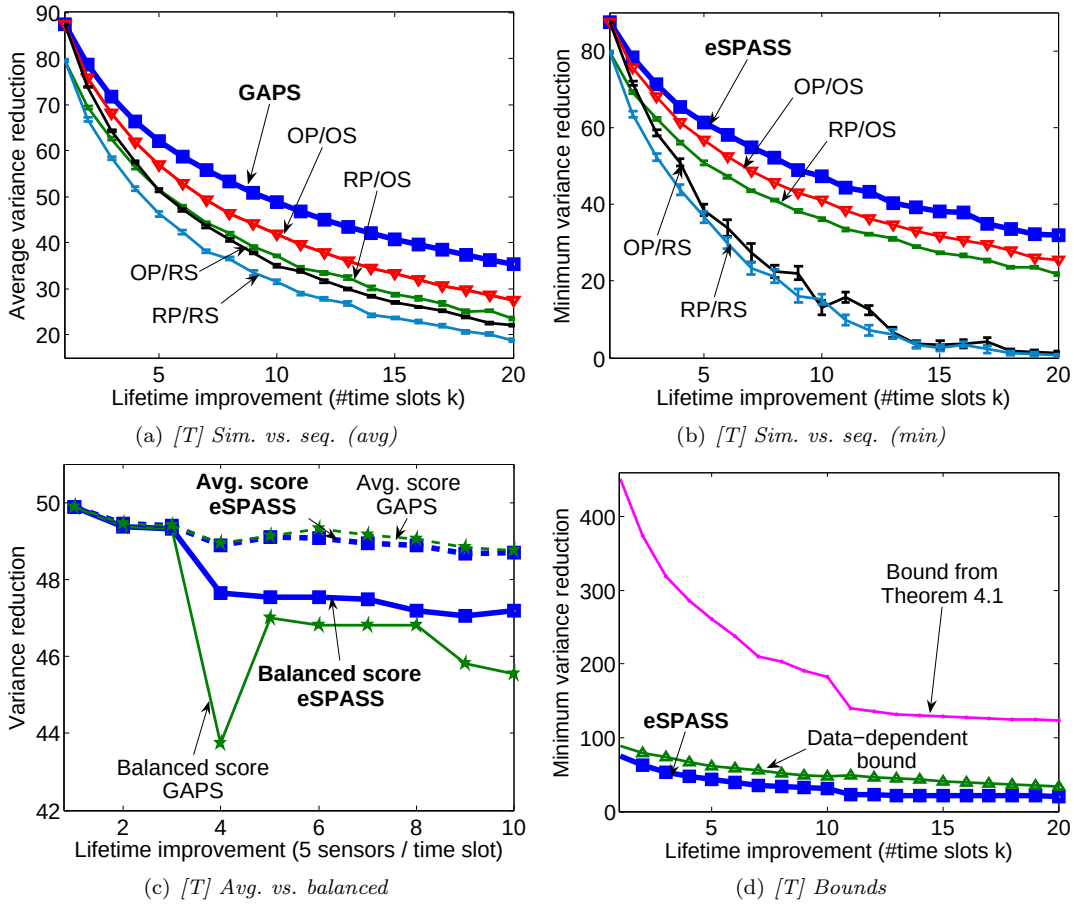
Figure 4: Placements and schedules for the traffic data. (a) Stage-wise approach, (b) SPASS solution.

As wireless sensors displace existing loop technologies, it is desirable to place as few sensors as possible, without trading off too much sensing quality. To achieve these goals in a principled manner, historical loop data from the current deployment should be used. eSPASS provides a solution which can balance these conflicting requirements, by combining scheduling and placement: more sensors are placed initially, still keeping road closures at a minimum, and scheduling is used to extend the lifetime of the network, keeping sensing quality balanced. In this section we explore this solution and compare eSPASS to other simultaneous placement and scheduling solutions.

Simultaneous vs. stage-wise optimization. In our first experiment, we study the benefit of simultaneously placing and scheduling sensors. For varying numbers m of sensors and k of time slots, we use different strategies to find k disjoint sets $\mathcal{A}_1, \ldots, \mathcal{A}_k$, where $\mathcal{A}_i$ is the sensors active at time slot i. We compare the simultaneous placement and schedule (optimized using eSPASS and GAPS) with solutions obtained by first placing sensors at a fixed set of locations, and then scheduling them. We consider both optimized and random sensor placements, followed by optimized and random scheduling, amounting to four stage-wise strategies. For random placements and schedules, we report the mean and standard error over 20 random trials.

Figure 5(a) presents the performance of the five strategies when optimizing the average-case performance, for a fixed number of $m = 50$ sensors and a number of time slots k varying from 1 to 20. GAPS performs best, followed by the stage-wise optimized placement and schedule (OP/OS). Of the two strategies where one component (either the placement or the schedule) is randomized (OP/RS and RP/OS), for small numbers (≤ 3) of time slots OP/RS performs slightly better, and for large numbers of time slots (≥ 10), RP/OS performs slightly better. The completely randomized solution performs significantly worse.

Figure 5(b) presents the same results when optimizing the balanced criterion. eSPASS outperforms the stage-wise strategies and the completely randomized strategy RP/RS performs worst as expected. Interestingly, for the balanced criterion, OP/RS performs drastically worse than RP/OS for $k \geq 4$ time slots. We hypothesize this to be due to the fact that a poor random placement can more easily be compensated for by using a good schedule than vice versa: When partitioning a sensor placement of 50 sensors randomly into a large number of time slots, it is fairly likely that at least one of the timeslots exhibits poor performance, hence leading to a poor balanced score. This insight also suggests that the larger the intended improvement in network lifetime (number of time slots), the more important it is to optimize for a balanced schedule.



(a) *[T] Sim. vs. seq. (avg)*

(b) *[T] Sim. vs. seq. (min)*

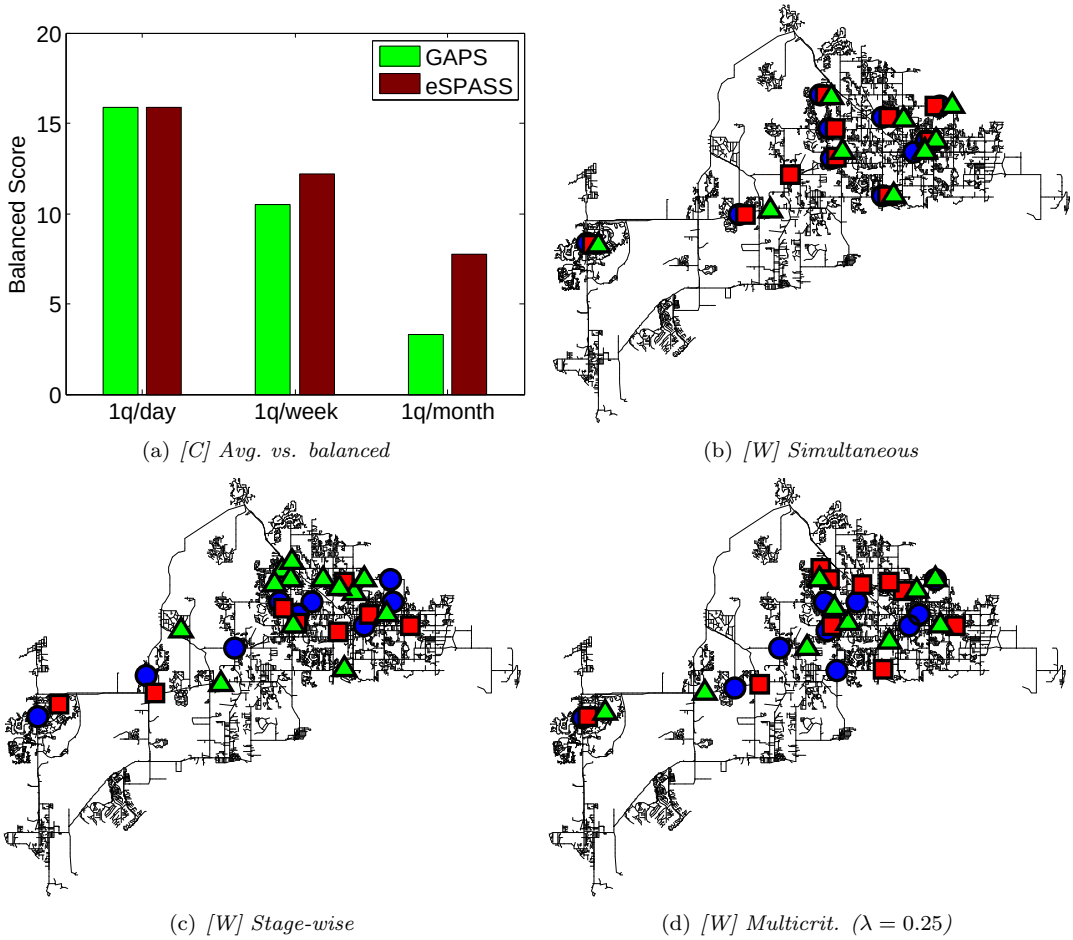
(c) *[T] Avg. vs. balanced*

(d) *[T] Bounds*

Figure 5: Results for traffic monitoring [T]. (a,b) compare simultaneous placement and scheduling to stage-wise strategies on (a) average-case and (b) balanced performance ($m = 50$, k varies). (c) compare average-case and balanced (using eSPASS) performance. (d) "Online" (data-dependent) bounds show that the eSPASS solutions are closer to optimal than the factor 6 "offline" bound from Theorem 4.1 suggests.

To summarize this analysis, we see that simultaneous placement and scheduling drastically outperforms the stage-wise strategies. For example, if we place 50 sensors at random, and then use eSPASS to schedule them into 4 time slots, we achieve an estimated minimum reduction in Mean Squared error by 58%. If we first optimize the placement and then use eSPASS for scheduling, we can achieve the same amount of variance reduction by scheduling 6 time slots (hence obtaining a 50% increase in network lifetime). If instead of stage-wise optimization we simultaneously optimize the placement and the schedule using eSPASS, we can obtain the same variance reduction by scheduling 8 time slots, hence an increase in network lifetime by 100%.

Average vs. balanced performance. We have seen that simultaneously placing and scheduling can drastically outperform stage-wise strategies, for both the average-case and the balanced objective. But which of the objectives should we use? In order to gain insight into this question, we performed the following experiment. For varying k and m, we obtain solutions to the SPASS problem using both the eSPASS and the GAPS algorithm. We then evaluate the respective solutions both using the average-case and the balanced criterion. Figure 5(c) presents the results of this experiment for k varying from 1 to 10, and fixed ratio of 5 sensors per time slot. As expected, eS-



(a) *[C] Avg. vs. balanced*

(b) *[W] Simultaneous*

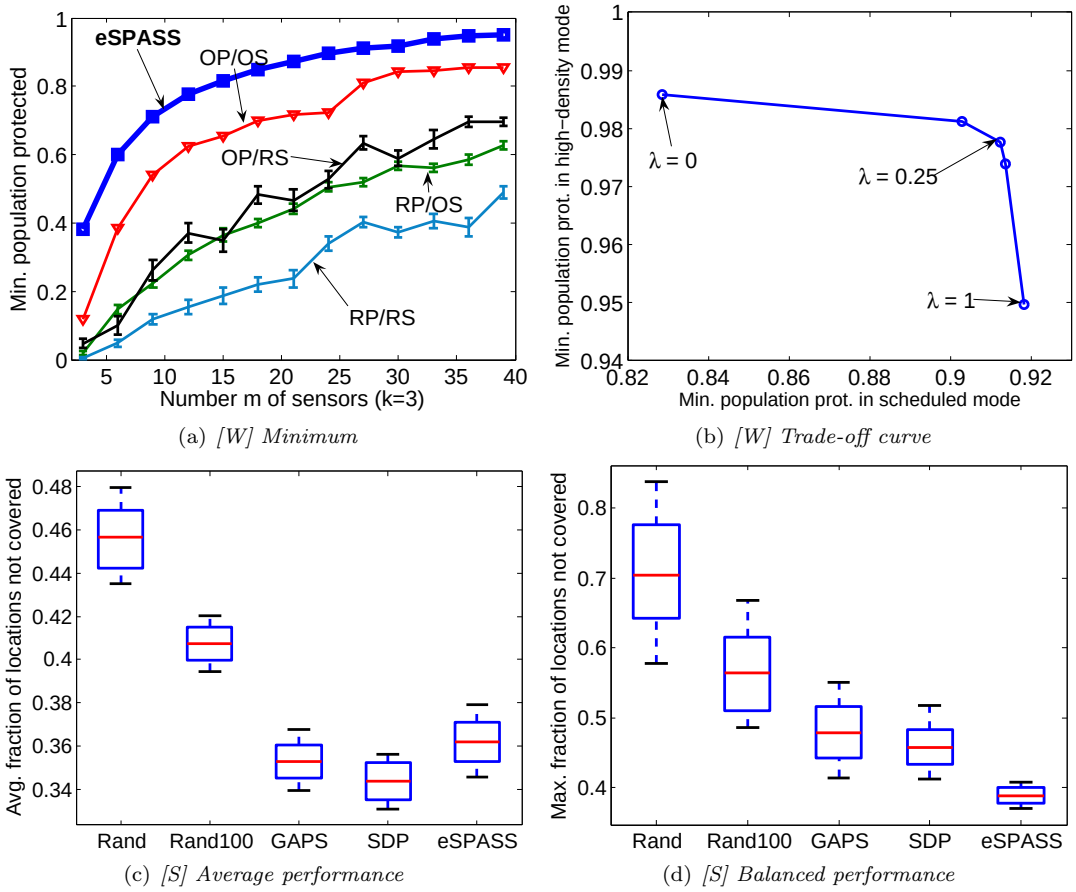
(c) *[W] Stage-wise*

(d) *[W] Multicrit.* ($\lambda = 0.25$)

Figure 6: (a) Results for community sensing [C]. When querying each car only once each week (using the eSPASS schedule), the sensing quality is only 23% lower than when querying every day. (b-d) Example placements and schedules for water networks [W].

PASS outperforms GAPS with respect to the balanced criterion, and GAPS outperforms eSPASS according to the average-case criterion. However, while eSPASS achieves average-case score very close to the solution obtained by GAPS, the balanced score of the GAPS solutions are far worse than those obtained by eSPASS. Hence, optimizing for the balanced criterion performs well for the average case, but not vice versa.

Online bounds. In order to see how close the solutions obtained by eSPASS are to the optimal solution, we also compute the bounds from Theorem 4.5. Figure 5(d) presents the bounds on the maximum variance reduction achievable when placing 50 sensors and partitioning them into an increasing number of groups. We plot both the factor 6 bound due to Theorem 4.1, as well as the data-dependent bound due to Theorem 4.5. We can see that the data dependent bounds are much tighter. For example, if we partition the sensors into 2 groups, our solution is at least 78% of optimum, for 5 groups it is at least 70% of optimum (rather than the 17% of Theorem 4.1).



(a) *[W] Minimum* (b) *[W] Trade-off curve*

(c) *[S] Average performance* (d) *[S] Balanced performance*

Figure 7: (a,b) Contamination detection in water networks [W]. (a) compares simultaneous and stage-wise solutions. (b) power/accuracy tradeoff curve with strong knee. (c,d) compares eSPASS with existing solutions on synthetic data [S].

6.2 Case study II: Community Sensing

While the static deployment of sensors has become an important means for monitoring traffic on highways and arterial roads, due to high deployment and maintenance cost it is difficult to extend sensor coverage to urban, side-streets. However, in order to optimize road-network utilization, accurate estimates of side-street conditions is essential.

Instead of (or in addition to) statically deploying sensors, a promising approach, studied by Krause et al. (2008a), would be to utilize cars as traffic sensors: An increasing number of vehicles nowadays are equipped with GPS and Personal Navigation Devices, which can accurately localize a car on a road network. Furthermore, these devices are becoming connected to wireless networks, using, e.g., GPRS or Edge connectivity, through which they could report their location and speed. Hence, in principle, it is possible to access accurate sensor data through the network of cars.

However, there are significant challenges when dealing with such a network of non-centrally owned sensors. While users may generally consider sharing their sensor data, they have reasonable concerns about their privacy. Krause et al. (2008a) provide methods for *community sensing*, describing strategies for selectively querying a community sensor network while maintaining preferences about privacy. They demonstrated how the selective querying of such a community sensor network can be modeled as the problem of optimizing a monotonic submodular sensing quality function that consid-

ers demand based on road usage. Preferences about privacy map to constraints in the optimization problem.

One basic preference that needs to be supported is the preference that each user is queried at most once in a specified time interval (e.g., queried at most once each week or month). Let $\mathcal{V}$ be the set of users subscribing to the community sensing service. In order to query each user at most one in k time steps, one strategy would be to partition the users into k sets $\mathcal{A}_1, \ldots, \mathcal{A}_k$, such that at time step t, users $\mathcal{A}_t$ are queried. In order to obtain continuously high performance of the monitoring service, we want to make sure that the performance $F(\mathcal{A}_t)$ is maximized simultaneously over all time steps. This is exactly an instance of the SPASS problem.

In order to evaluate the performance of the eSPASS algorithm, we used the experimental setup of Krause et al. (2008a), using real traffic data from 534 detector loops deployed underneath highways, GPS traces from 85 volunteer drivers and demand data based on directions generated in response to requests to a traffic prediction and route planning prototype named ClearFlow, developed at Microsoft Research[5]. Details about the data sets are described by Krause et al. (2008a). Based on this experimental setup, we compare the performance of the eSPASS and GAPS algorithms (latter of which was described as a candidate scheduling approach by Krause et al. (2008a)). Using each algorithm, we partition the users into 7 respective 31 sets (i.e., querying each user at most once each week respective month, both of which are possible options for privacy preferences). We then evaluate the performance based on the worst case prediction error over all test time steps. Figure 6(a) presents the results of this experiment. We can see that the eSPASS solutions outperform the GAPS solutions. When partitioning into 31 sets, the worst prediction performance of eSPASS is more than twice as good than the worst prediction performance of GAPS. Most importantly, this experiment shows that, using eSPASS for scheduling, one can obtain a very high balanced performance, even when querying each individual car only very infrequently. For example, when querying each car only once each week, the balanced sensing quality is only 23% lower than that obtained by the privacy-intrusive continuous (daily) querying. Even if querying only once each month (i.e., a factor 31 more infrequently), the balanced performance is only reduced by approximately a factor of 2. These results indicate that, using eSPASS, even stringent preferences about privacy can be met without losing much prediction accuracy.

6.3 Case study III: Contamination detection

Consider a city water distribution network, delivering water to households via a system of pipes, pumps and junctions. Accidental or malicious intrusions can cause contaminants to spread over the network, and we want to select a few locations (pipe junctions) to install sensors, in order to detect these contaminations as quickly as possible. In August 2006, the Battle of Water Sensor Networks (BWSN) (et al., 2008) was organized as an international challenge to find the best sensor placements for a real (but anonymized) metropolitan water distribution network, consisting of 12,527 nodes. In this challenge, a set of intrusion scenarios is specified, and for each scenario a realistic simulator provided by the EPA (Rossman, 1999) is used to simulate the spread of the contaminant for a 48 hour period. An intrusion is considered detected when one selected node shows positive contaminant concentration.

The goal of BWSN was to minimize impact measures, such as the expected population affected, which is calculated using a realistic disease model. Krause et al. (2008b) showed that the function $F(\mathcal{A})$ which measures the expected population protected by placing sensors at location $\mathcal{A}$ is a monotonic submodular function. Water quality sondes can operate for a fairly long amount of time on battery power. For example, the YSI 6600 Sonde can sample 15 water quality parameters every 15 minutes for 75 days. However, for the long-term feasibility it is desirable to considerably improve this battery lifetime by sensor scheduling. On the other hand, high sampling rates are

[5]The ClearFlow research system, available only to users within Microsoft Corporation, was the prototype for the Clearflow context-sensitive routing service now available publicly for North American cities at http://maps.live.com.

desirable to ensure rapid response to possible contaminations. For a security-critical sensing task such as protecting drinking water from contamination, it is important to obtain balanced, uniformly good detection performance over time. In addition, deployment and maintenance cost restrict the number of sensors that can be deployed. Hence, the problem of deploying battery powered sensors for drinking water quality monitoring is another natural instance of the SPASS problem.

We reproduce the experimental setup detailed in (Krause et al., 2008b). However, instead of only optimizing for the sensor placement, we simultaneously optimize for placement and schedule using the eSPASS algorithm. Figure 7(a) compares eSPASS with the stage-wise approaches. For each algorithm, we report the population protected by placing sensors, normalized by the maximum protection achievable when placing sensors at every node in the network.

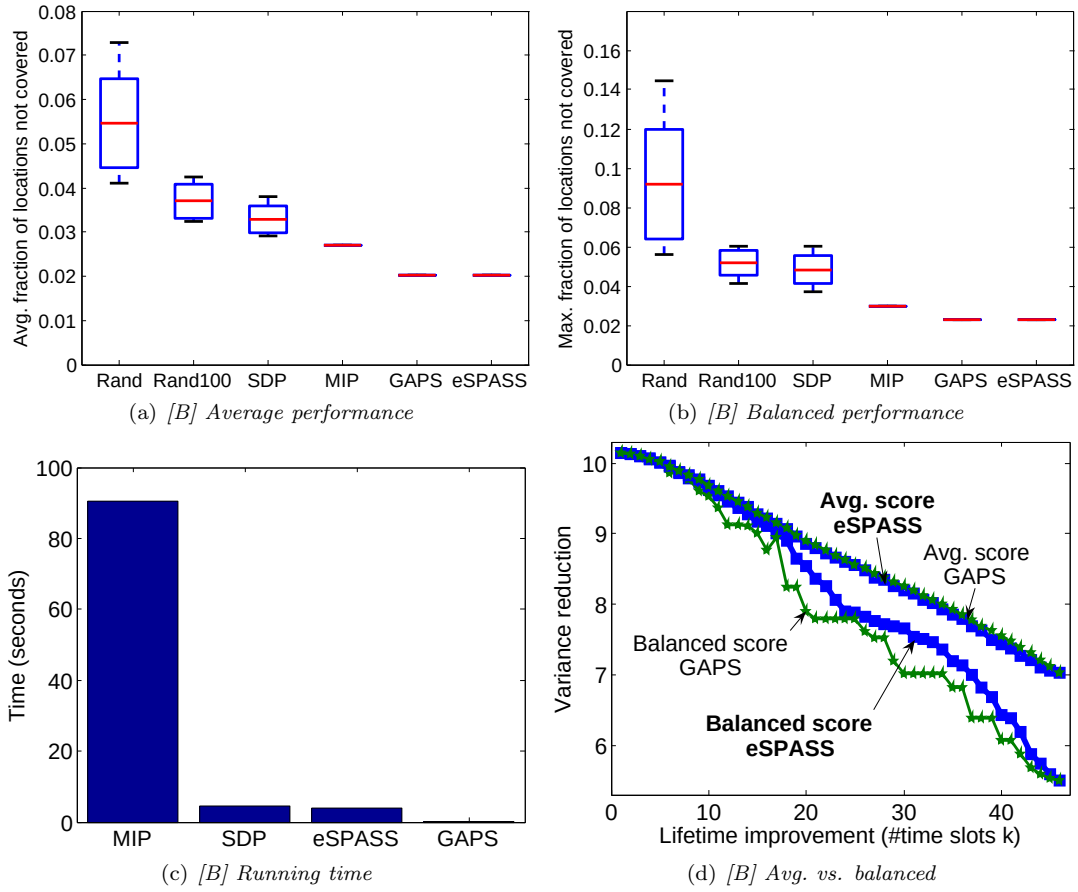
Simultaneous vs. stage-wise optimization. eSPASS obtains drastically improved performance when compared to the stage-wise approaches. For example, when scheduling 3 time slots, in order to obtain 85% protection, eSPASS requires 18 sensors. The fully optimized stage-wise approach (OP/OS) requires twice the number of sensors. When placing 36 sensors, the stage-wise approach leaves 3 times more population unprotected as compared to the simultaneous eSPASS solution with the same number of sensors. eSPASS solved this large scale optimization task ($n = 12,527$, $k = 3$, $m = 30$) in 26 minutes using our MATLAB implementation.

Trading off power and accuracy. We also applied our modified eSPASS algorithm in order to trade off scheduled mode and high density mode performance. For a fixed number of $m = 30$ sensors and $k = 3$ time slots, we solve Problem (4) for values of λ varying from 0 to 1. For each value of λ, we obtain a different solution, and plot the normalized expected population protected (higher is better) both in scheduled- and in high-density mode in Figure 7(b). We can see that this trade-off curve exhibits a prominent knee, where solutions are obtained that perform nearly optimally with respect to both criteria. Figures 6(b), 6(c) and 6(d) show the placements and schedules obtained for $\lambda = 1$ (i.e., ignoring the high-density sensing quality), $\lambda = 0$ (ignoring the schedule, effectively performing a stage-wise approach) and a value $\lambda = 0.25$ (from the knee in the trade-off curve) respectively. Note how the solution for $\lambda = 1$ clusters the sensors closely together, obtaining three very similar placements $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ for each time slot (similar as in Figure 2). The solution for $\lambda = 0$ spreads out the sensors more, having to leave, e.g., the Western part of the network uncovered in the time slot indicated by the green triangle. The multicriterion solution ($\lambda = 0.25$) is a compromise between the former two solutions: The sensors are still clustered together, but also spread out more – the Western part of the network can be covered in this solution.

6.4 Case study IV: Multiple weblog coverage

The fourth case study is in a very different application domain, an online information network rather than a physically connected sensor network. In Leskovec et al. (2007), Leskovec et al. studied the problem of selecting informative weblogs to read on the Internet. Our approach is based on the intuitive notion of an information cascade: A blogger writes a posting, and, after some time, other blogs link to it. An information cascade is a directed acyclic graph of vertices (each vertex corresponds to a posting at some blog), where edges are annotated by the time difference between the postings. Based on this notion of an information cascade, we would like to select blogs, that detect big cascades (containing many nodes) as early as possible (i.e., we want to learn about an important event before most other readers). In Leskovec et al. (2007) it is shown how one can formalize this intuition using a monotonic submodular function F that measures the informativeness of a subset $\mathcal{A}$ of all blogs $\mathcal{V}$. Optimizing the submodular function F leads to a small set $\mathcal{A}$ of blogs that "covers" most cascades. Since blogs from different topics (such as politics, religion, tech news, etc.) participate in different cascades, the solution set $\mathcal{A}$ will typically contain very *diverse* blogs. Similar blogs that overlap significantly suffer from diminishing returns.

While such a notion of coverage is intuitive, sometimes multiple coverage is desired: We might be interested in selecting "multiple representatives" from each topic, whereby we allow some (limited)



(a) *[B] Average performance*

(b) *[B] Balanced performance*

(c) *[B] Running time*

(d) *[B] Avg. vs. balanced*

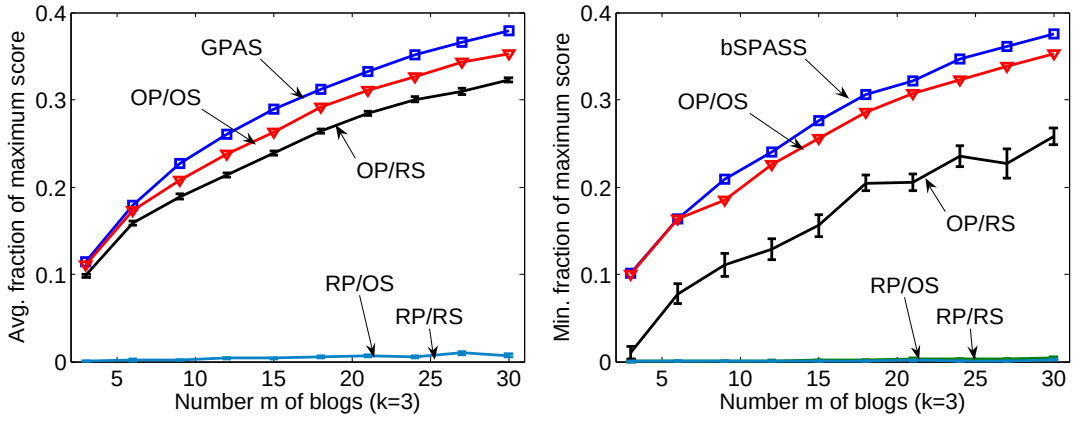
Figure 8: Results on temperature data from Intel Research Berkeley [B]. (a,b) compares eSPASS with existing solutions. (c) compares running time. (d) compares average-case and balanced performance.

amount overlap among the covered cascades. For example, there might be political blogs that participate in the same cascades (discussions), and hence have a lot of overlap, but it is nevertheless desirable to select multiple blogs as each of them presents their own perspective.

We can formulate this requirement as an instance of the SPASS problem: We would like to select multiple, disjoint sets of blogs, that each provide a large amount of information.

We reproduce the experimental setup of Leskovec et al. (2007), that is based on 45,000 blogs, as well as one million posts during 2006. This data set contains 16,000 cascades in which at least 10 blogs participate. We use the *population affected* objective function described in detail in Leskovec et al. (2007). Based on this experimental setup, we apply eSPASS to partition an increasing number of blogs into three groups (to allow up to three-fold coverage). We optimize the problem using the eSPASS algorithm. As comparison, we first select the set of blogs (using the approach described in Leskovec et al. (2007)), and then use the eSPASS algorithm to partition the selected set into three groups ($k = 3$). We also compare against the other stage-wise approaches as done in Section 6.1.

Figure 9 presents the results of this experiment. It shows that blog selections using the eSPASS algorithm obtain 10% better performance than the stage-wise approach. Further note that eSPASS is able to solve the large problem instance with $n = 45,000$ within 26 minutes.



(a) *[W] Average* (b) *[W] Minimum*

Figure 9: Multiple coverage for blogs

6.5 Comparison with existing techniques

We also compare ESPASS with several existing algorithms. Since the existing algorithms apply to the scheduling problem only, we call ESPASS with $m = |\mathcal{V}|$ (i.e., allow it to select all sensors).

Set covering. Most existing algorithms for sensor scheduling assume that sensors are associated with a fixed sensing region that can be perfectly observed by the sensor (*c.f.*, Abrams et al. (2004); Deshpande et al. ('08)). In this setting, we associate with each location $s \in \mathcal{A}$ a set $\mathcal{R}_s \subseteq \mathcal{V}$ of locations that can be monitored by the sensor, and define the sensing quality $F(\mathcal{A}) = |\bigcup_{s\in\mathcal{A}} \mathcal{R}_s|$ to be the total area covered by all sensors. Since set coverage is an example of a monotonic submodular function, we can use ESPASS to optimize it.

We compare ESPASS to the greedy approach by Abrams et al. (2004), as well as the the approach by Deshpande et al. ('08) that relies on solving a semidefinite program (SDP). We use the synthetic experimental setup defined by Deshpande et al. ('08) to compare the approaches. A set of n sensors is used to cover M regions. Each sensor s is associated with a set $\mathcal{R}_s$ of regions it covers. The objective is to divide the n sensors into k groups (buckets), such that the minimum or the average number regions covered by each group is maximized.

For the SDP by Deshpande et al. ('08), we solve the SDP using SeDuMi to get a distribution over possible schedules, and then pick the best solution out of 100 random samples drawn from this distribution. For the random assignment approach (Rand100) of Abrams et al. (2004), we sample 100 random schedules and pick the best one. In addition, we run the GAPS and the ESPASS algorithms. We apply those four algorithms to 50 random set cover instances as defined by Deshpande et al. ('08): for each sensor, a uniform random integer r between 3 and 5 is chosen, and then the first r regions from a random permutation of the set of M regions is assigned to that sensor. The sensor network size is $n = 20$, the number of desired groups $k = 5$ and the number of regions is $M = 50$.

Figure 7(c) presents the average performance of the four approaches. In this setting, the SDP performs best, closely followed by GAPS and ESPASS. Figure 7(d) presents the balanced performance of the four approaches. Here, ESPASS significantly outperforms both the SDP and the GAPS solution.

Building monitoring. As argued in the introduction, for complex spatial monitoring problems, the sensing region assumption is unrealistic, and we would rather like to optimize prediction accuracy directly. The approach by Koushanfary et al. (2006) is designed to schedule sensors under constraints on the prediction accuracy. Their approach, given a required prediction accuracy, constructs a prediction graph that encodes which sensors can predict which other sensors. They then solve

a domatic partitioning problem, selecting a maximal number of disjoint subsets that can predict all other sensors with the desired accuracy. In order to determine the domatic partitioning, their algorithm relies on the solution of a Mixed Integer Program (MIP). However, solving MIPs is NP-hard in general, and unfortunately, we were not able to scale their approach to the traffic data application. Instead, we use data from 46 temperature sensors deployed at Intel Research, Berkeley (*c.f.*, Deshpande et al. (2004)).

On this smaller data set, we first apply the MIP for domatic partitioning, with a specified accuracy constraint. The MIP was very sensitive with respect to this accuracy constraint. For just slightly too small values of ε, the MIP returned a trivial solution consisting of only a single set. For slightly too large values, the MIP had to consider partitions into a large number of possible time slots, increasing the size of the MIP such that the solver ran out of memory. Requiring that sensors can predict each other with a Root Mean Squared (RMS) error of 1.25 Kelvin leads to a selection of $m = 19$ sensors, partitioned into $k = 3$ time slots. Using this setting for m and k, we run the GAPS and eSPASS algorithms, which happen to return the same solution for this example. In order to compare these solutions with the SDP and random selection from the previous section, we apply them to the prediction graph induced by the required prediction accuracy. We first randomly select 19 locations, and then partition them into 3 groups using the SDP and Rand100 approach, respectively. As a baseline, we randomly select 3 groups totaling 19 sensors (Rand). For these randomized techniques, we report the distribution over 20 trials. All approaches are evaluated based on the variance reduction objective function.

Figure 8(a) presents the result for optimizing the average variance reduction, and Figure 8(b) for the minimum variance reduction. In both settings, GAPS and eSPASS perform best, obtaining 23% less remaining maximum variance when compared to the MIP solution of Koushanfary et al. (2006). Furthermore, using YalMIP in Matlab, solving the MIP requires 95 seconds, as compared to 4 seconds for the SDP and 3.8 seconds for eSPASS (Figure 8(c)). Even though the MIP returns an optimum solution for the domatic partition of the prediction graph, eSPASS performs better since it uses the fact that the combination of multiple sensors can lead to better prediction accuracy than only using single sensors for prediction. Even the best out of 20 random trials for the SDP performs worse than the MIP, due to the approximate nature of the algorithm and the random selection of the initial 19 sensors. The Rand100 approach does performs only slightly (not significantly) worse than the SDP based approach.

7 Related Work

In the context of wireless sensor networks, where sensor nodes have limited battery and can hence only enable a small number of measurements, optimally placing and scheduling sensors is of key importance.

Sensor Placement. Many approaches for optimizing sensor placements assume that sensors have a fixed region (Hochbaum and Maas, 1985; Gonzalez-Banos and Latombe, 2001; Bai et al., 2006). These regions are usually convex or even circular. Furthermore, it is assumed that everything within this region can be perfectly observed, and everything outside cannot be measured by the sensors. For complex applications such as traffic monitoring however, such assumptions are unrealistic, and the direct optimization of prediction accuracy is desired. The problem of selecting observations for monitoring spatial phenomena has been investigated extensively in geostatistics (*c.f.*, Cressie (1991) for an overview), and more generally (Bayesian) experimental design (*c.f.*, Chaloner and Verdinelli (1995)). Submodularity has been used to analyze algorithms for placing a fixed set of sensors (Krause et al., 2007). These approaches however only consider the sensor placement problem, and not the scheduling aspect.

Sensor Scheduling. The problem of deciding when to selectively turn on sensors in order to conserve power was first discussed by Slijepcevic and Potkonjak (2001) and Zhao et al. (2002).

Typically, it is assumed that sensors are associated with a fixed sensing region, and a spatial domain needs to be covered by the regions associated with the selected sensors. Abrams et al. (2004) presents an efficient approximation algorithm with theoretical guarantees for this problem. Deshpande et al. ('08) presents an approach for this problem based on semidefinite programming (SDP), handling more general constraints and providing tighter approximations. They also provide a randomized rounding based approach for scheduling under the balanced objective (which they call min-coverage (time)). However, in contrast to eSPASS (when specialized to scheduling) their algorithm requires to relax the constraint that each sensor location can only be selected once. Also, their guarantee only holds with high probability, whereas eSPASS is deterministic. The approaches described above do not apply to the problem of optimizing sensor schedules for more complex sensing quality functions such as, e.g., the increase in prediction accuracy and other sensing quality functions considered in this paper. To address these shortcomings, Koushanfary et al. (2006) developed an approach for sensor scheduling that guarantees a specified prediction accuracy based on a regression model. However, their approach relies on the solution of a Mixed Integer Program, which is intractable in general. Zhao et al. (2002) proposed heuristics for selectively querying nodes in a sensor network in order to reduce the entropy of the prediction. Unlike the algorithms presented in this paper, their approaches do not have any performance guarantees.

Submodular optimization. The problem of maximizing a submodular function subject to a matroid constraint, of which Problem (1) is an instance, has been studied by Fisher et al. (1978), who proved that the greedy algorithm gives a factor 2 approximation. Recently, Vondrak (2008) showed that a more complex algorithm achieves a (1-1/e) approximation to this problem. Note that this algorithm could be applied to the Problem (1) instead of GAPS. Furthermore note that using this algorithm as a subroutine, the analysis of eSPASS can be improved to give a $\frac{e-1}{3e} \geq \frac{1}{5}$ guarantee. Comparing this algorithm is an interesting direction for future work. A related version of optimization Problem (2), where for each time step t a different function F_t is used has been studied in the context of combinatorial allocation problems. Ponnuswami and Khot (2007) present an algorithm that guarantees a $1/(2k-1)$ approximation. For the special case where the objective functions F_t are additive (modular), Asadpour and Saberi (2007) developed an algorithm that guarantees an improved $\Omega\left(\frac{1}{\sqrt{k}\log^3 k}\right)$ approximation. Both algorithms however only apply for the scheduling setting (i.e., they require that $m = |\mathcal{V}|$). Furthermore, note that the approximation performance of these algorithms very quickly decreases with k, in contrast to our eSPASS approach that provides an approximation guarantee that is independent of the number of time steps. Krause et al. ('08) consider the problem of robust maximization of submodular functions: Given a collection of submodular functions, $F_1, \ldots, F_m$, they want to find a set $|\mathcal{A}| \leq k$ that maximizes $\min_i F_i(\mathcal{A})$. While this problem appears related to the SPASS problem, where we want to maximize $\min_i F(\mathcal{A}_i)$, the solution techniques and results are very different. Firstly, there is a strong conceptual difference: In robust submodular optimization, a single set $\mathcal{A}$ is sought that maximizes multiple functions $F_1, \ldots, F_m$, whereas in SPASS, a collection of sets $\mathcal{A}_1, \ldots, \mathcal{A}_k$ is sought that each perform well with respect to a single function F. Hence, the two problem formulations address very different optimization tasks. Second, while both algorithms exploit the fact that truncation preserves submodularity, each contains unique algorithmic elements. Lastly, the performance guarantees vary drastically: the robust submodular optimization problem does not admit any approximation (and requires the relaxation of the constraint that $|\mathcal{A}| \leq k$), whereas for the SPASS problem, eSPASS obtains a constant-factor 6 approximation.

8 Conclusions

When deploying sensor networks for monitoring tasks, both placing and scheduling the sensors are of key importance, in order to ensure informative measurements and long deployment lifetime. Traditionally, the problems of sensor placement and scheduling have been considered separately

from each other. In this paper, we presented an efficient algorithm, eSPASS, that simultaneously optimizes the sensor placement and the schedule. We considered both the setting where the average-case performance over time is optimized, as well as the balanced setting, where uniformly good performance is required. Such balanced performance is crucial for security-critical applications such as contamination detection. Our results indicate that optimizing for balanced performance often yields good average-case performance, but not necessarily vice versa. We proved that our eSPASS algorithm provides a constant factor 6 approximation to the optimal balanced solution. To the best of our knowledge, eSPASS is the first algorithm that provides strong guarantees for this problem, partly resolving an open problem raised by Abrams et al. (2004). Furthermore, our algorithm applies to any setting where the sensing quality function is submodular, which allows to address complex sensing tasks where one intends to optimize prediction accuracy or optimizes detection performance.

We also considered complex sensor placement scenarios, where the deployed sensor network must be able to function well both in a scheduled, low power mode, but also in a high accuracy mode, where all sensors are activated simultaneously. We developed an algorithm, mcSPASS, that directly optimizes this power-accuracy tradeoff. Our results show that mcSPASS yields solutions which perform near-optimally with respect to both the scheduled and the high-density performance.

We extensively evaluated our approach on several real-world sensing case studies, including traffic and building monitoring as well as contamination detection in metropolitan area drinking water networks. When applied to the simpler special case of sensor scheduling (i.e., ignoring the placement aspect), eSPASS outperforms existing sensor scheduling algorithms on standard data sets. For the more complex, general case, our algorithm performs provably near-optimal (as demonstrated by tight, data-dependent bounds). Our results show that, for fixed deployment budget, drastic improvements in sensor network lifetime can be achieved by simultaneously optimizing the placement and the schedule, as compared to the traditional, stage-wise approach. For example, for traffic prediction, eSPASS achieves a 33% improvement in network lifetime compared to the setting where placement and scheduled are optimized separately, and a 100% improvement when compared to the traditional setting where sensors are first randomly deployed and then optimally scheduled.

We believe that the results presented in this paper present an important step towards understanding the deployment and maintenance of real world sensor networks.

Acknowledgments

We would like to thank Dr. Eric Horvitz and Dr. John Krumm who made available to us the experimental setup and data used in the community sensing project at Microsoft Research. We would also like to thank Prof. Pravin Varaiya and Dr. Sinem Coleri Ergen for valuable discussions, Sensys Networks for providing technical advice on their sensor, as well as Khalid El-Arini and Joseph Gonzalez for helpful comments on the paper. This work was partially supported by NSF Grants No. CNS-0509383, CNS-0625518, CCF-0448095, CCF-0729022, and a gift from Intel. Anupam Gupta and Carlos Guestrin were partly supported by Alfred P. Sloan Fellowships, Carlos Guestrin by an IBM Faculty Fellowship and an ONR Young Investigator Award N00014-08-1-0752 (2008-2011). Andreas Krause was partly supported by a Microsoft Research Graduate Fellowship. Ram Rajagopal was supported by ARO-MURI UCSC-W911NF-05-1-0246-VA-09/05.

References

Z. Abrams, A. Goel, and S. Plotkin. Set k-cover algorithms for energy efficient monitoring in wireless sensor networks. In *IPSN*, 2004.

A. Asadpour and A. Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In *STOC*, pages 114–121, New York, NY, USA, 2007. ACM Press. ISBN 978-1-59593-631-8. doi: http://doi.acm.org/10.1145/1250790.1250808.

X. Bai, S. Kumar, Z. Yun, D. Xuan, and T. H. Lai. Deploying wireless sensors to achieve both coverage and connectivity. In *ACM MobiHoc*, Florence, Italy, 2006.

UC Berkeley. http://pems.eecs.berkeley.edu.

K. Chaloner and I. Verdinelli. Bayesian experimental design: A review. *Stat. Sci.*, 10(3):273–304, Aug. 1995. ISSN 08834237.

N. A. C. Cressie. *Statistics for Spatial Data*. Wiley, 1991.

A. Das and D. Kempe. Algorithms for subset selection in linear regression. In *STOC*, 2008.

A. Deshpande, S. Khuller, A. Malekian, and M. Toossi. Energy efficient monitoring in sensor networks. In *Latin*, '08.

A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

A. Ostfeld et al. The battle of the water sensor networks (BWSN): A design challenge for engineers and algorithms. *To appear in J. Wat. Res. Plan. Mgmt.*, 2008.

M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for maximizing submodular set functions - ii. *Math. Prog. Study*, (8):73–87, 1978.

T. Fujito. Approximation algorithms for submodular set cover with applications. *TIEICE*, 2000.

H. H. Gonzalez-Banos and J. Latombe. A randomized art-gallery algorithm for sensor placement. In *Proc. 17th ACM Symp. Comp. Geom.*, pages 232–240, 2001.

A. Haoui, R. Kavaler, and P. Varaiya. Wireless magnetic sensors for traffic surveillance. *Transportation Research C*, 16(3):294–306, 2008.

D. S. Hochbaum and W. Maas. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32:130–136, 1985.

F. Koushanfary, N. Taft, and M. Potkonjak. Sleeping coordination for comprehensive sensing using isotonic regression and domatic partitions. In *Infocom*, 2006.

A. Krause, B. McMahan, C. Guestrin, and A. Gupta. Selecting observations against adversarial objectives. In *NIPS*, '08.

A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. In *JMLR*, 2007.

A. Krause, E. Horvitz, A. Kansal, and F. Zhao. Towards community sensing. In *IPSN*, 2008a.

A. Krause, J. Leskovec, C. Guestrin, J. VanBriesen, and C. Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *J. Wat. Res. Plan. Mgmt.*, 136(6), 2008b.

J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *KDD*, 2007.

A.K. Ponnuswami and S. Khot. Approximation algorithms for the max-min allocation problem. In *APPROX*, 2007.

L.A. Rossman. The epanet programmer's toolkit for analysis of water distribution systems. In *Annual Water Resources Planning and Management Conference*, 1999.

S. Slijepcevic and M. Potkonjak. Power efficient organization of wireless sensor networks. In *ICC*, 2001.

J. Vondrak. Optimal approximation for the submodular welfare problem in the value oracle model. In *STOC*, 2008.

J.L. Williams, J.W. Fisher III, and A.S. Willsky. Performance guarantees for information theoretic active inference. In *AISTATS*, 2007.

F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing*, 19(2):61–72, 2002.

A Proofs

Proof of Theorem 3.1. Define a new ground set $\mathcal{V}' = \mathcal{V} \times \{1, \ldots, k\}$, and a new function

$$F'(\mathcal{A}') = \sum_{t=1}^{k} F(\{s : (s,t) \in \mathcal{A}'\}).$$

F' is monotonic and submodular. Let

$$\mathcal{I} = \{\mathcal{A}' \subseteq \mathcal{V}' : |\mathcal{A}'| \leq m \wedge (\nexists s, i \neq j : (s,i) \in \mathcal{A}' \wedge (s,j) \in \mathcal{A}')\},$$

i.e., $\mathcal{I}$ is the collection of subsets $\mathcal{A}' \subseteq \mathcal{V}'$ that do not contain two pairs (s,i) and (s,j) of elements for which $i \neq j$. It can be shown that $\mathcal{I}$ form independent sets of a matroid (*c.f.*, Fisher et al. (1978)). Note that there is a one-to-one correspondence between sets $\mathcal{A}'$ and feasible solutions $\mathcal{A}_1, \ldots, \mathcal{A}_k$ to the SPASS Problem (1), and furthermore, the corresponding solutions have the same value. Hence, the SPASS problem is equivalent to solving

$$\mathcal{A}^* = \operatorname*{argmax}_{\mathcal{A}' \in \mathcal{I}} F'(\mathcal{A}').$$

As Fisher et al. (1978) proved, the greedy algorithm GAPS is guaranteed to obtain a solution that has at least 1/2 of the optimal value. □

Proof of Lemma 4.2. Consider an optimal allocation $\mathcal{T}_1, \ldots, \mathcal{T}_m$. Let $\mathcal{B}_{opt}$ be the set $\mathcal{B}_{opt} = \{i : \mathcal{T}_i \text{ contains a big element}\}$. Throw away all buckets (and elements) $\mathcal{B}_{opt}$. Now, in order to achieve score c, the optimal solution has to fill $m - |\mathcal{B}_{opt}|$ buckets with small elements (even from a reduced set of small elements, those not thrown away) and still achieve score c on each of those buckets. This solution is in fact an optimal solution achieving score c on the new problem instance (since we will use at least as many big elements and throw away at least as many buckets). □

Proof of Lemma 4.3. Suppose $\mathcal{A}_i$ is a bucket for which $F_c(\mathcal{A}_i) \geq 3\beta c$. Now, $\mathcal{A}_i = \{a_1, \ldots, a_l\}$, and $F_c(\{a_i\}) \leq \beta c$. Choose ℓ such that $F_c(\{a_1, \ldots, a_{\ell-1}\}) < \beta c$ and $F_c(\{a_1, \ldots, a_\ell\}) \geq \beta c$. Let $\Delta = \{a_1, \ldots, a_\ell\}$.

Due to monotonicity $F_c(\mathcal{A}_j \cup \Delta) \geq F_c(\Delta) \geq \beta c$. It remains to show that $F_c(\mathcal{A}_i \setminus \Delta) \geq F_c(\mathcal{A}_i) - 2\beta c$. Suppose that $F_c(\mathcal{A}_i \setminus \Delta) < F_c(\mathcal{A}_i) - 2\beta c$. Let $\mathcal{B} = \mathcal{A}_i \setminus \Delta$. Then

$$F_c(\mathcal{B} \cup \Delta) - F_c(\mathcal{B}) > 2\beta c.$$

But

$$F_c(\Delta) \leq F_c(\{a_1, \ldots, a_{\ell-1}\}) + F_c(\{a_\ell\}) < 2\beta c,$$

due to submodularity of F_c, and the fact that a_ℓ is a small element. Hence

$$F_c(\mathcal{B} \cup \Delta) - F_c(\mathcal{B}) > F_c(\Delta) - F_c(\emptyset),$$

i.e., adding Δ to $\mathcal{B}$ helps more than adding Δ to the empty set, contradicting submodularity of F_c. □

Proof of Lemma 4.4. To simplify notation, w.l.o.g. let us assume that $c = 1$. Since the optimal balanced performance for $F_c = F_1$ is 1, the optimal average-case performance for F_1 is 1 as well. The GAPS algorithm obtains an allocation $\mathcal{A}$ that is a fraction α of optimal. Hence, it holds that $\sum_i F_1(\mathcal{A}_i) \geq \alpha k$. We call $\sum_i F_1(\mathcal{A}_i)$ the "mass" of the allocation $\mathcal{A}$.

How many unsatisfied buckets can there maximally be? Let γ denote the fraction of unsatisfied buckets. We know that

$$k\gamma\beta + k(1-\gamma) \geq \alpha k,$$

since the maximal γ is achieved if all the satisfied buckets are completely full (containing mass $k(1-\gamma)$), and the unsatisfied buckets are as full as possible without being satisfied (hence containing mass less than $k\gamma\beta$). Hence it follows that

$$\gamma \leq \frac{1-\alpha}{1-\beta}.$$

Now consider the mass R distributed over the satisfied buckets. We know that

$$R \geq \alpha k - \gamma k\beta \geq k\frac{\alpha(1-\beta) - \beta(1-\alpha)}{1-\beta},$$

and the worst case is assumed under equality.

The first reallocation move is possible if

$$\frac{R}{k(1-\gamma)} \geq 3\beta,$$

since, if the average remaining mass over all $(1-\gamma)k$ satisfied buckets is 3β, then there must be at least one bucket to which the move can be applied. Since each reallocation move reduces the mass R by at most 2β (as proved by Lemma 4.3), and since we need γk moves to fill all unsatisfied buckets, it suffices to require that

$$\begin{aligned}
&\frac{R - 2\gamma k\beta}{k(1-\gamma)} \geq 3\beta \\
\Leftrightarrow &R - 2\gamma k\beta \geq 3\beta k - 3\beta\gamma k \\
\Leftrightarrow &R \geq 3\beta k - \beta\gamma k
\end{aligned}$$

Hence, a sufficient condition for β such that enough moves can be performed to fill all unsatisfied buckets is

$$\begin{aligned}
&\frac{\alpha(1-\beta) - \beta(1-\alpha)}{1-\beta} \geq 3\beta - \beta\frac{1-\alpha}{1-\beta} \\
\Leftarrow &3\beta^2 + (-3-\alpha)\beta + \alpha \geq 0 \\
\Leftarrow &\beta \leq \alpha/3,
\end{aligned}$$

by solving the quadratic equation for β and ignoring the infeasible solutions $\beta \geq 1$. Now, since β is going to be our approximation factor, we want to maximize β subject to the above constraint, and hence choose $\beta = \alpha/3$. □

Proof of Theorem 4.1. The proof immediately follows from the analysis in Section 4.2. For the running time, notice that, in each binary search iteration, the greedy algorithm requires at most kmn function evaluations, and the reallocation step requires at most $k^2m \leq kmn$ evaluations. The binary search terminates after $\mathcal{O}(1 + \log_2 F(\mathcal{V}))$ iterations, assuming integrality of F. □

Proof of Theorem 4.5. Let $\mathcal{A}_i$ be the candidate solution for time slot i, and $\mathcal{B}_i = \{b_1, \ldots, b_{n_i}\}$ an optimal solution for time slot i. Due to monotonicity and submodularity, it holds that

$$F(\mathcal{A}_i) \leq F(\mathcal{A}_i \cup \mathcal{B}_i) \leq F(\mathcal{A}_i) + \sum_{j=1}^{n_i} \delta_{i,b_j}.$$

Hence, the optimal value of Problem (2) is upper bounded by the optimal solution to the following integer program:

$$\begin{aligned} &\max_{\lambda_{i,s},c} c \text{ s.t.} \\ &\qquad c \leq F(\mathcal{A}_i) + \sum_s \lambda_{i,s}\delta_{i,s} \text{ for all } i \\ &\sum_i \lambda_{i,s} \leq 1 \text{ for all } s \text{ and } \sum_{i,s} \lambda_{i,s} \leq m \text{ and } \lambda_{i,s} \in \{0,1\}, \end{aligned}$$

since any integer solution $\lambda_{i,s}$ corresponds to a possible feasible partition $\mathcal{B}'_1, \ldots, \mathcal{B}'_k$. The linear program in Theorem 4.5 is the linear programming relaxation to the above integer program. □

Algorithm mcGAPS (F, $\mathcal{B}$, $\mathcal{V}$, k, m, c, λ)
$\mathcal{A}_t \leftarrow \emptyset$ for all t; $\mathcal{A} \leftarrow \emptyset$;
for $i = 1$ ***to*** m **do**
  foreach $s \in \mathcal{V} \setminus \mathcal{A}$, $1 \leq t \leq k$ **do**
    $\delta_{t,s} \leftarrow \lambda F_c(\mathcal{A}_t \cup \{s\}) + (1-\lambda)F(\mathcal{A} \cup \mathcal{B} \cup \{s\})$;
  end
  $(t^*, s^*) \leftarrow \text{argmax}_{t,s}\, \delta_{t,s}$;
  $\mathcal{A}_{t^*} \leftarrow \mathcal{A}_{t^*} \cup \{s^*\}$; $\mathcal{A} \leftarrow \mathcal{A} \cup \{s^*\}$;
end

Algorithm 3: The greedy average-case placement and scheduling (mcGAPS) algorithm.

Proof Sketch of Theorem 5.1. We will modify eSPASS in the following way. The modified algorithm, mcSPASS (see the pseudo code in Algorithm 4), will "guess" (binary search for) the value c^* attained by the optimal solution $\mathcal{A}^*$. It will then guess (search for) the number ℓ of large elements used in the optimal solution, where we redefine "large" as $F(\{s\}) \geq \frac{c^*}{8}$. For such a guess, mcSPASS will first greedily select the ℓ large elements (according to F), giving a set $\mathcal{A}_G$. It will then set these large elements aside, and continue on the small elements. Define the function

$$G(\mathcal{A}_1, \ldots, \mathcal{A}_k) = (1-\lambda)(F(\mathcal{A}_G \cup \mathcal{A}) - F(\mathcal{A}_G)) + \frac{\lambda}{m-\ell} \sum_i \min\{F(\mathcal{A}_i), c\}$$

where $\mathcal{A} = \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_k$. mcSPASS will greedily maximize G on the partition matroid (similarly to using GAPS). Suppose c^* is the optimal value $c^* = \widehat{F}_\lambda(\mathcal{A}^*)$. Then the greedy procedure will find a solution $\mathcal{A}'$ such that $G(\mathcal{A}') + (1-\lambda)F(\mathcal{A}_G) \geq \frac{1}{2}c^*$ (since greedy selection of big elements followed by greedy selection of small elements amounts to the "local" greedy optimization over a partition matroid as analyzed by Fisher et al. (1978)).

```
Algorithm mcSPASS (F, 𝒱, k, m, ε, λ)
c_min ← 0; c_max ← F(𝒱); β ← 1/8;
while c_max − c_min ≥ ε do
    for ℓ = 0 to k do
        c ← (c_max + c_min)/2;
1       ℬ ← {s ∈ 𝒱 : F_c({s}) ≥ βc};
        if |ℬ| < ℓ then break;
        𝒜_big ← ∅;
        for i = 1 to ℓ do
            𝒜_big ← argmax_{s∈ℬ\𝒜_big} F(𝒜_big ∪ {s});
            𝒜_{k−i+1} ← {s};
        end
        k′ ← k − ℓ;
        if k′ = 0 then 𝒜_{best,ℓ} ← (𝒜_1, ..., 𝒜_k);
          continue;
        𝒱′ ← 𝒱 \ 𝒜_big; m′ ← m − ℓ;
2       𝒜_{1:k′} ←mcGAPS (F, 𝒜_big, 𝒱′, k′, m′, c, λ);
3       if Σ_t F(𝒜_t) < k′c/2 then c_max ← c; continue;
        else
4           while ∃i, j ≤ k′: F_c(𝒜_j) ≤ βc, F_c(𝒜_i) ≥ 3βc do
                foreach s ∈ 𝒜_i do
                    𝒜_j ← 𝒜_j ∪ {s}; 𝒜_i ← 𝒜_i \ {s};
                    if F_c(𝒜_j) ≥ βc then break;
                end
            end
            𝒜_{best,ℓ} ← (𝒜_1, ..., 𝒜_k);
        end
    end
    ℓ* ← argmax_ℓ F̂_λ(𝒜_{best,ℓ});
    c_min ← c; 𝒜_best ← 𝒜_{best,ℓ*};
end
```

Algorithm 4: The mcSPASS algorithm for simultaneously optimizing scheduled and high-density performance.

Now, at least one of $(1-\lambda)F(\mathcal{A}_G \cup \mathcal{A}') \geq c^*/8$, or $\frac{\lambda}{m-\ell}\sum_i F(\mathcal{A}'_i) \geq \frac{3c^*}{8}$, since $G(\mathcal{A}') + (1-\lambda)F(\mathcal{A}_G) = (1-\lambda)F(\mathcal{A}_G \cup \mathcal{A}') + \lambda\frac{1}{l-m}\sum_i F(\mathcal{A}'_i)$. In former case we do not need to reallocate and set $\mathcal{A}_R = \mathcal{A}'$. In latter case, we use the reallocation procedure, and arrive at a solution $\mathcal{A}_R$ where all buckets are satisfied (since $\mathcal{A}'$ contains only small elements), i.e., $\min_i F(\mathcal{A}_{R,i}) \geq \frac{3c^*}{3\cdot 8} = c^*/8$.

Now, $\mathcal{A}_R \cup \mathcal{A}_G$ is a feasible solution to the multicriterion SPASS problem, with

$$\begin{aligned}\widehat{F}_\lambda(\mathcal{A}_R \cup \mathcal{A}_G) &= (1-\lambda)F(\mathcal{A}_G) + G(\mathcal{A}_R) \\ &= (1-\lambda)F(\mathcal{A}_G \cup \mathcal{A}_R) + \lambda \min_i F(\mathcal{A}_{R,i}) \geq c^*/8.\end{aligned}$$

□

Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Carnegie Mellon