OPTIMIZING RESOURCE ALLOCATION UNDER CONSTRAINTS

by

MATTHEW P. JOHNSON

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2010

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Amotz Bar-Noy

Date — Chair of Examining Committee

Theodore Brown

Date — Executive Officer

Thomas La Porta

Janos Pach

Rohit Parikh

Supervision Committee

THE CITY UNIVERSITY OF NEW YORK

Acknowledgements

I have many to be thankful for–friends, colleagues, and others.

I could not have been more fortunate in advisors than I was with Prof. Amotz Bar-Noy. He taught me much that I know about computer science, research, and much else. He was unreasonably kind, patient, and generous, in all sorts of ways.

I thank all my advisors from earlier stages of my academic career, each of whom has influenced me, including Prof. Michael Grossberg, who brought me to CUNY.

I thank the members of my disseration committee for their guidance and support, including Prof. Tom La Porta, with whom I've worked very closely.

I thank Ted Brown and Lina Garcia for helpful administrative support at all points, and Andres Varon for advice on preparing this document.

I thank all my many coauthors, including Amotz Bar-Noy, Fangfei Chen, Yi Feng, Tom La Porta, Ou Liu, Hosam Rowaihy, Deniz Sarioz, and Chai Wah Wu. The published papers underlying this thesis benefited enormously from all of them.

Finally, I wish to thank a few others (left unnamed here) for moral support during my years of graduate study.

The research on energy charge scheduling was supported by grants from the NSF (grant number 0332596) and the New York State Office of Science, Technology and Academic Research. The research on sensor assignment and coverage was sponsored by US Army Research Laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001.

Abstract

OPTIMIZING RESOURCE ALLOCATION UNDER CONSTRAINTS

by

Matthew P. Johnson

Advisor: Professor Amotz Bar-Noy

The fundamental concept of resource allocation–assigning resources to tasks, and so on–gives rise to very broad families of problems, with instances in many domains, applications, disciplines. In the proposed thesis, we will study a series of assignment-oriented resource allocation problems modeling real-world situations, from the point of view of online and approximation algorithms. The complexities and character of these problems vary greatly. Separate from the problem definitions, we will also examine how the difficulty changes as we vary modalities such as online v. offline, distributed v. centralized, linear v. poly-time, thus mapping out a lattice of problem settings. For each problem and setting, we seek efficient algorithms of the appropriate kind (e.g., exact, approximation, competitive, distributed, ...). Three classes of problems examined in particular will be sensor-mission matching, battery charge scheduling, and geometric sensor coverage.

Table of Contents

Chapter 1

Introduction

Problems of resource allocation are fundamental to many realms of human endeavor, in a possibly unlimited range of domains, from management and operations research, to aid and development, indeed to the self-interested life choices made by an individual *Homo economicus*. These form a very broad class of problems involving optimization under constraints. (Resource allocation is a hammer that pounds many nails.) The objective function may have to do with maximizing profit or quality of information, or minimizing cost, delay, or the strain on infrastructure; the constraints may be computational in nature (e.g., when solving the problem exactly is intractable) or a feature of the problem description (e.g., operating in an online or distributed environment). Restricting to two disjoint sets of nodes (jobs and machines, or clients and servers) results in myriad *matching* or *assignment* problems (see [7] and references therein).

Consider, for example, personnel management, one of the principle duties of a corporate manager. Tasks come in different shapes and sizes, and workers come equipped with different abilities. This raises the question of how best to divvy up the work among employees. In practice, there are many additional complications. Workers may reside in different

locales, or be available at different times. For tasks requiring multiple workers, this greatly increases the complexity of the problem (although as recently seen in out-sourcing, these problems can sometimes cancel one another out).

Or consider ad hoc sensor networks used in rescue applications. At an earthquake location, we might drop wireless sensors from the air, in order to search for survivals in specific locations, measure air quality, etc. Given a set of available sensors and a set of targets to observe or *missions* to accomplish, we face the task of assigning sensors to missions. Further complicating matters, a mission may be a request for a certain amount or quality of information, of a certain kind (e.g., video or temperature), describing a certain location. A given sensor may offer different missions varying amount of information (because of geometry, for example), or none at all (because of differing information types). Separate from a mission's requirements, missions may vary in importance. Already we find a large number of potential problems in this for this one application.

1.1 Approximation Algorithms

Since the development of the theory of NP-completeness [45] in the 1960s and 1970s, and the realization that many important optimization problems apparently [5] cannot be solved efficiently by classical computers, researchers began studying approximation algorithms for hard problems, i.e., algorithms that produce solutions which while not necessarily optimal are guaranteed to be multiplicatively close to optimal. Applying dimension-reduction, the bulk of the world in this area can be partitioned in two ways: combinatorial versus LP-based and constant-approximation versus PTAS.

Definition 1.1.1. *For a given problem instance I, let $|OPT_I|$ indicate an optimal solution to*

a given problem instance, and let $|ALG_I|$ indicate a corresponding approximate solution. We will say that an algorithm is a c-approximation *for $c \geq 1$ if $\frac{|OPT_I|}{|ALG_I|} \leq c$ for every problem instance I.*[1]

1.2 Online Algorithms

Online algorithms are designed for situations in which we must act in real time, with limited or no knowledge of the future, and with prior actions irrevocable. A canonical example is the ski rental problem [40], in which renting skis costs (say) $10 per day and buying skis costs $100. (Each time we go skiing we have the rent-or-buy choice, if and until we buy; once we buy, there no further decision is made.) For our future skiing schedule, we want to spend a little money on skis as possible. Given full knowledge of the skiing schedule, this is an easy decision–buy iff we ski at least 10 times. In the online problem, however, we do not know what the future will bring. After we ski for the first time, we might ski frequently or never again. And while the future is often like the past [59] (though which aspect of the past [48]?), we refuse to take advantage of induction here. Our goal is an algorithm that guarantees near-optimal expenditures, without knowledge of the future, *come what may*.

Definition 1.2.1. *For a given problem instance I, let $|OPT_I|$ indicate an optimal solution to a given problem instance, and let $|ALG_I|$ indicate a corresponding solution given by an online algorithm. We will say that an algorithm is* c-competitive *for $c \geq 1$ if $\frac{|OPT_I|}{|ALG_I|} \leq c$ for every problem instance I.*

This problem may sound impossible to solve, but in fact it's easy to show that the following algorithm is 2-competitive: *rent until you've spent as much as it costs to buy,*

[1] *The definition of approximation, as well as the definition of competitiveness below, can be weakened to allow additive error in addition to multiplicative error, i.e., a corresponding asymptotic definition.*

and then buy. After all, if we ski fewer at most 10 times, then the performance is optimal; if we ski more than 10 times, then the performance ratio is at most 2.

The motivation for competitiveness, originally introduced by Sleator & Tarjan [91], is that we would like our online (maximization) algorithm to be (nearly) as good as any possible online algorithm, but there's apparently no way to know what the best possible online algorithm could guarantee. What the best possible offline algorithm could guarantee is in a sense obvious: the optimal solution. Hence by guaranteeing that we're close to the optimal offline solution, we ensure that we are as close to the performance of the best possible online algorithm:

$$|OPT|/c \geq |OPT_{on}| \geq |ALG|$$

An analogous definition and analysis can be given for minimization problems.

1.3 Scope

We will study a series of feasibility and optimization problems that model contemporary real-world situations. Most of the problems we consider are generalizations of maximum-matching, max-flow, or both. Like in many other areas of computer science, however, it turns out that this small set of building blocks suffices to produce a problem space of enormous complexity. As we will see, different assumptions or restrictions will give rise to dramatically varying computational complexities, algorithms, and approaches. We prove the complexities of these problems (both feasibility and optimization) and develop efficient algorithms to solve, optimize, or approximate them, as appropriate.

Some of the problems will be online in nature, meaning that we receive the problem

instance over time but must respond to each portion immediately [91, 24, 70]. Online problems have long been studied through competitive analysis, where we compare an online algorithm's solution quality to that of the optimal offline algorithm and seek algorithms for which this ratio is bounded by a constant. In fact, we can apply the approach of competitive analysis to many other modalities: distributed v. centralized, dynamic v. static, linear v. poly v. unbounded-time. (Of particular interest are dynamic settings, in which the size of a job e.g. can change over time.) In this way, we may render commensurable otherwise distinct and unrelated restrictions.

1.4 Organization

This thesis is divided into three parts. Chapters 2-4 study problems of battery charge scheduling, primarily from the point of view of online algorithms. Chapters 5-8 turn to sensor networks, in which we study the allocation of individual sensors to competing missions, including problem variants both abstract and applied. Finally, Chapters 9-12 are concerned with sensor coverage. In those chapters, we study coverage of discrete targets as well as continuous areas.

Chapter 2

Battery Charge Scheduling

There is increasing interest in saving fuel costs by use of renewable energy sources such as wind and solar power. Although such sources are highly desirable, and the power they provide is in a sense free, the typical disadvantage is unreliability: availability depends e.g. on weather conditions (it is not "dispatchable" on demand). Many companies seek to build efficient systems to gather such energy when available and store it, perhaps in modified form, for future use [97].

On the other hand, power companies charge some high-consumption clients not just for the total amount of power consumed, but also for how quickly they consume it. Within the billing period (typically a month), the client is charged for the amount of energy used (*usage charge*, in kWh) and for the maximum amount requested over time (*peak charge*, in kW). [1] If demands are given as a sequence $(d_1, d_2, \ldots, d_n)$, then the total bill is of the form $c_1 \sum_i d_i + c_2 \max_i\{d_i\}$ (for some constants $c_1, c_2 > 0$), i.e., a weighted sum of the total usage and the maximum usage. (In practice, the discrete timeslots may be 30-minute averages [3].) This means that a client who powers a 100kW piece of machinery for one

[1] In fact, some billing models are more complex.

hour and then uses no more energy for the rest of the month would be charged more than a client who uses a total of 100kWh spread evenly over the course of the month. Since the per-unit cost for peak charges may be on the order of 100 times the per-unit cost for total usage [4], [2] this difference can be significant. Indeed, this is borne out in our experiments.

This suggests a second use for the battery: to store *purchased* energy for future use. Indeed, at least one start-up company[3] is currently marketing such a battery-based system intended to reduce peak energy charges. In such a system, a battery is placed between the power company and a high-consumption client site, (such as a large office building or factory) in order to smooth power requests and shave the peak. The client site will charge to the battery when demand is low and discharge when demand is high. Spikes in the demand curve can thus be rendered consistent with a relatively flat level of supplied power. The result is a lower cost for the client and a more manageable request curve for the provider.

It is interesting to note that a battery system may actually *raise* energy usage, since there may be energy loss due to inefficiency in AC/DC conversion. This loss may be as much as 33% of the amount charged. Serving peak requests during periods of high demand is a difficult and expensive task for the power company, however, and the event of a black-out inflicts high societal costs. While a battery system may involve higher total energy requests, it may benefit the system as a whole by easing the strain of peak demands. Combined with alternative energy sources such as solar panels, the system could even lower the net commercial power usage. Alternative energy sources are typically low-cost but unreliable, since they depend on external events such as the weather. With a battery, this energy can be stored until needed.

[2]The Orlando Utilities Commission website (`www.ouc.com/account/rates/electric-comm.htm`), for example, quotes rates of 6.388 cents per kWh ("energy charge") and $6.50 per kW ("demand charge").

[3]Gaia Power Technologies: `gaiapowertech.com`.

We may generalize this problem of minimaxing the request to any resource which is *tenable* in the sense that it may be obtained early and stored until needed. For example, companies frequently face shortages of popular products: "Plentiful supply [of Xboxes] would be possible only if Microsoft made millions of consoles in advance and stored them without releasing them, or if it built vast production lines that only ran for a few weeks– both economically unwise strategies," a recent news story asserted [52]. A producer could smooth the product production curve by increasing production and warehousing supply until future sales. But when should the producer "charge" and "discharge"? (In some domains, there may also be an unpredictable level of volunteer help.) A third application is the scheduling of jobs composed of generic work-units that may be done in advance. Although the problem is very general, we will use the language of energy and batteries for concreteness. Many features of this production problem, including uncertainty in future demand, a bounded warehouse size and a cost for storage in the warehouse have analogs in the battery problem.

In the online version of our problem, the essential choice faced at each timeslot is whether (and by how much) to invest in the future or to cash in a prior investment. The investment in our setting is a request for more energy than is needed at the time. If the algorithm only asks for the minimum required, then it is vulnerable to spikes in demand; if it asks for much more energy than it needs, then the greater request could itself introduce a new, higher peak. The strictness of the problem lies in the fact that the cost is not cumulative: we want *every* request to be low.

The offline version is easy to solve in its most basic form, but becomes more difficult when realistic complications are included. Our online algorithms will be based on the optimal offline algorithms.

First, we will give H_n-competitive algorithms for the online lossless setting and matching lower bounds on competitiveness. (These algorithms are in fact only partially online since they depend on having the maximum demand D revealed in advance. Lacking this information, no non-trivial competitiveness is possible.) Those algorithms assume perfectly efficient batteries, however, and will fail if run on realistic, lossy batteries. Second, we will adapt these algorithms to the lossy setting, testing them on both synthetic and actual customer usage data. Moreover, we test more aggressive, heuristic algorithms, as well as algorithms that accept predictions, with error, of future demands. We also consider new settings and objective functions, such as total cost. Finally, we provide factor-revealing LPs, which we use to provide quasi-empirical evidence of the competitiveness of the lossy algorithms.

The goal in the minimax work-scheduling problem [60] is to minimize the maximum amount of work done in any timeslot over a finite time-horizon. Our online problem is related to a previously studied special case of this in which jobs with deadlines are assigned online. In that problem, all work must be done by deadline but cannot be begun until assigned. While the problems differ in important respects (see [15]), the objectives are similar. Indeed, while the α-policy of [60] performs α *times the maximum per-unit-timeslot amount of work that OPT would have done, when running on the partial input received so far*, many of our algorithms ensure that the savings at each point is a multiple of the optimal savings so far.

This part presents results previously appearing in [16, 14]. This work was supported by grants from the NSF (grant number 0332596) and the New York State Office of Science, Technology and Academic Research. We thank Deniz Sariöz and Ted Brown for useful discussions. We also thank Ib Olsen of Gaia for providing the Starbucks dataset.

Chapter 3

Lossless Batteries

3.1 Introduction

In this chapter, we define and present results for the batter charge problem, assuming that batteries are perfectly efficient in the sense defined below. At each timeslot the system meets an energy demand through a combination of a new request, an unreliable amount of *free source* energy (e.g. solar or wind power), and previously received energy. The added piece of infrastructure is the *battery*, which can store surplus energy for future use. More generally, the demands could represent required amounts of energy, water, or any other *tenable* resource which can be obtained in advance and held until needed. In a feasible solution, each demand must be supplied on time, through a combination of newly requested energy, energy withdrawn from the battery, and free source. The goal is to minimize the maximum request.

We will consider batteries with and without a bounded capacity, and with or without a percentage loss in charging due to inefficiency. In the online version of this problem, the algorithm must determine each request without knowledge of future demands or free

source availability, with the goal of maximizing the amount by which the peak is reduced.

We will give efficient optimal algorithms for the offline problem, with and without a bounded battery. Central to our analysis is a mathematical property we call a *generalized average*. Our fastest offline algorithms for the lossy battery settings compute a series of generalized averages with the aid of balanced binary search trees. We also show how to find the optimal offline battery size, given the requirement that the final battery level equals the initial battery level. Finally, we will give efficient H_n-competitive algorithms assuming the peak *effective* demand is revealed in advance, and provide matching lower bounds.

Background. Experimental work applying variations of these online algorithms to settings lacking provable guarantees was recently presented [14]. The present chapter focuses on settings that allow guaranteed competitiveness.

There is a wide literature on commodity production, storage, warehousing, and supply-chain management (see e.g. [72, 106, 47, 84]). More specifically, there are a number of inventory problems based on the Economic Lot Sizing model [41], in which demand levels for a product vary over a discrete finite time-horizon and are known in advance. A feasible solution in these problems must obtain sufficient supply through production (sometimes construed as ordering) or through other methods, in order to meet each of the demands on time, while observing certain constraints. The nature of solution quality varies by formulation.

One such inventory problem is Single-Item Lot-Sizing, in which sufficient supplies must be ordered to satisfy each demand, while minimizing the total cost of ordering charges and holding charges. The ordering charge consists of a fixed charge per order plus a charge linear in order size. The holding charge for inventory is per-unit and per-timeslot. There is a tradeoff between these incentives since fixed ordering charges encourage large orders while

holding charges discourage them. Wagner & Whitin [96] showed in 1958 that this problem can be solved in polynomial time. Under the assumption of *non-speculative costs*, in which case orders should always be placed as late as possible, the problem can be solved in linear time. Such "speculative" behavior, however, is the very motivation of our problem. There are many lot-sizing variations, including constant-capacity models that limit the amount ordered per timeslot. (See [84] and references therein.) Our offline problem differs in that our objective is minimizing this constant capacity (for orders), subject to a bound on *inventory* size, and we have no inventory charge.

Another related inventory problem is Capacity and Subcontracting with Inventory (CSI) [9], which incorporates trade-offs between production costs, subcontracting costs, holding costs, and the cost for maximum per-unit-timeslot production capacity. The goal in that problem is to choose a production capacity and a feasible production/ subcontracting schedule that together minimize total cost, whereas in our problem choosing a production capacity, subject to storage constraints, is the essential task.

In the minimax work-scheduling problem [60], the goal is to minimize the maximum amount of work done in any timeslot over a finite time-horizon. Our online problem is related to a previously studied special case in which jobs with deadlines are assigned online. In that problem, all work must be done by deadline but cannot be begun until assigned. Subject to these restrictions, the goal is to minimize the maximum work done in any timeslot. While the optimization goal is the same, our online problem differs in two respects. First, each job for us is due immediately when assigned. Second, we *are* allowed to do work (request and store energy) in advance. One online algorithm for the jobs-by-deadlines problem is the α*-policy* [60]: at each timeslot, the amount of work done is α *times the maximum per-unit-timeslot amount of work that OPT would have done, when running on the partial*

input received so far. One of our online algorithms adopts a similar strategy.

Contributions. We introduce a novel scheduling problem and solve several versions optimally with efficient combinatorial algorithms. We solve the offline problem for two kinds of batteries: unbounded battery in $O(n)$ time and bounded in $O(n^2)$. Separately, we show how to find the optimal offline battery size, for the setting in which the final battery level must equal the initial battery level. This is the smallest battery size that achieves the optimal peak. The online problem we study is very strict. A meta-strategy in many online problems is to balance expensive periods with cheap ones, so that the overall cost stays low [24]. The difficulty in our problem lies in its non-cumulative nature: we optimize for the max, not for the average. We show that several versions of the online problem have no algorithm with non-trivial competitive ratio (i.e., better than n or $\Omega(\sqrt{n})$). Given advanced knowledge of the peak demand D, however, we give H_n-competitive algorithms for batteries bounded and unbounded. Our fastest algorithm has $O(1)$ per-slot running-time. H_n is the (optimal) competitive ratio for both battery settings.

Examples. Although there is no constant-ratio competitive algorithm for unbounded n, our intended application in fact presumes a fixed time-horizon. If the billing period is one month, and peak charges are computed as 30-minute averages, then for this setting H_n is approximately 7.84. If we assume that the battery can fully recharge at night, so that each day can be treated as a separate time period, then for a 12-hour daytime time-horizon H_n is approximately 3.76.

3.2 Model and Preliminaries

Definition 3.2.1. *At each timeslot i, d_i is the demand, r_i is the request, b_i is the battery charge level* at the start *of the timeslot, and f_i is the amount of free source available. By*

$\hat{d}_i$ *we indicate the* effective demand $d_i - f_i$*. We sometimes refer to the sequence over time of one of these value types as a curve, e.g., the* demand curve*.* D *is the maximum effective demand* $\max_i\{\hat{d}_i\}$*, and* R *is the maximum request* $\max_i\{r_i\}$*.*

The problem instance comprises the demands, the free source curve, battery size B, initial charge b_1, and required final charge b_{n+1} (in the offline case). The problem solution consists of the request curve.

Definition 3.2.2. *Let* overflow *be the situation in which* $r_i + f_i - d_i > B - b_i$*, i.e., there is not enough room in the battery for the amount we want to charge. Let* underflow *be the situation in which* $d_i - r_i - f_i > b_i$*, i.e., there is not enough energy in the battery for the amount we want to discharge. Call an algorithm* feasible *if underflow never occurs. The goal of the problem is to minimize* R *(for competitiveness measures this is construed as maximizing* $D - R$*) while maintaining feasibility.*

In the absence of overflow/underflow, the battery level at timeslot i is simply $b_i = b_{i-1} + r_{i-1} + f_{i-1} - d_{i-1}$. It is forbidden for b_i to ever fall below 0. That is, the request r_i, the free source f_i, and the battery level b_i must sum to at least the demand d_i at each timeslot i. Notice that effective demand can be negative, which means that the battery may be charged (capacity allowing), even if the request is 0. We assume D, however, is strictly positive. Otherwise, the problem instance is essentially trivial. We use the following to simplify the problem statement:

Observation 3.2.1. If effective demands may be negative, then free source energy need not be explicitly considered.

As such, we set aside the notion of free source, and for the remainder of the chapter (simplifying notation) allow demand d_i to be negative.

In the energy application, battery capacity is measured in kWh, while instantaneous request is measured in kW. By discretizing we assume wlog that battery level, demand, and request values are expressed in common units. Peak charges are based linearly on the max request, which is what we optimize for. The battery can have a *maximum capacity* B or be unbounded. The problem may be online, offline, or in between; we consider the setting in which the peak demand D is revealed in advance, perhaps predicted from historical information.

Threshold algorithms: For a particular snapshot (d_i, r_i, b_i), demand d_i must be supplied through a combination of the request r_i and a change in battery $b_{i+1} - b_i$. This means that there are only three possible modes for each timeslot: request exactly the demand-free, request more than this and *charge* the difference, or request less and *discharge* the difference. We refer to our algorithms as *threshold algorithms*. Let $T_1, T_2, ..., T_n$ be a sequence of values. Then the following algorithm uses these as request thresholds:

for each timeslot i
    if $d_i < T_i$
        charge $min(B - b_i, T_i - d_i)$
    else
        discharge $d_i - T_i$

Intuitively, the algorithm amounts to the rule: *at each timeslot i, request an amount as near to T_i as the battery constraints will allow*. Our offline algorithms are *constant threshold* algorithms, with a fixed T; our online algorithms compute T_i dynamically for each timeslot i.

A constant-threshold algorithm is specifiable by a single number. In the online setting, predicting the *exact* optimal threshold from historical data suffices to solve the online algorithm optimally. A small overestimate of the threshold will merely raise the peak cost

correspondingly higher. Unfortunately, however, examples can be found in which even a small *underestimate* eventually depletes the battery before peak demand and thus produce no cost-savings at all.

The *offline* problem can be solved approximately, within additive error ϵ, through binary search for the minimum feasible constant threshold value T. Simply search the range $[0, D]$ for the largest value T for which the threshold algorithm has no underflow, in time $O(n \log \frac{D}{\epsilon})$. If the optimal peak reduction is $R-T$, then the algorithm's peak reduction will be at least $R - T - \epsilon$. It is straightforward to give a linear programming formulation of the *offline* problem; it can also be solved by generalized *parametric* max-flow [7]. Our interest here, however, is in efficient combinatorial optimal algorithms. Indeed, our combinatorial offline algorithms are significantly faster than these general techniques and lead naturally to our competitive online algorithms. Online algorithms based on such general techniques would be intractable for fine-grain timeslots.

3.3 Offline problem

We now find optimal algorithms for both battery settings. For unbounded, we assume the battery starts empty; for bounded, we assume the battery starts with amount B. For both, the final battery level is unspecified. We show below that these assumptions are made wlog. The two offline threshold functions, shown in Table 4.3, use the following definition:

Definition 3.3.1. *Let $\mu(j) = \frac{1}{j}\sum_{t=1}^{j} d_t$ be the mean demand of the prefix region $[1, j]$, and let $\hat{\mu}(k) = max_{1 \le j \le k}\mu(j)$ be the maximum mean among of the prefix regions up to k. Let $\rho(i, j) = \frac{-B+\sum_{t=i}^{j} d_t}{j-i+1}$ be the* density *of the region $[i, j]$ and $\hat{\rho}(k) = max_{1 \le i \le j \le k}\rho(i, j)$ be the maximum density among all subregions of $[1, k]$.*

Alg.	battery	threshold T_i	run-time
1.a	unbounded	$\hat{\mu}(n)$	$O(n)$
1.b	bounded	$\hat{\rho}(n)$	$O(n^2)$

Table 3.1: Threshold functions used for offline algorithm settings.

Bounded capacity changes the character of the offline problem. It suffices, however, to find the peak request made by the optimal algorithm, R_{opt}. Clearly $R_{opt} \geq D - B$, since the ideal case is that a width-one peak is reduced by size B. Of course, the peak region might be wider.

Theorem 3.3.1. *Algorithm* 1.a *(threshold $T_i = \hat{\mu}(n)$, for unbounded battery) and Algorithm* 1.b *(threshold $T_i = \hat{\rho}(n)$, for bounded battery) are optimal, feasible, and run in times $O(n)$ and $O(n^2)$, respectively.*

Proof. First, let the battery be unbounded. For any region $[1, j]$, the best we can hope for is that requests for all demands $d_1, ..., d_j$ can be spread evenly over the first j timeslots. Therefore the optimal threshold cannot be lower than the maximum $\mu(j)$, which is Algorithm *1.a*'s threshold. For feasibility, it suffices to show that after each time j, the battery level is nonnegative. But by time j, the total input to the system will be $j \cdot \hat{\mu}(n) \geq j \cdot \mu(j) = \sum_{t=1}^{j} d_j$, which is the total output to the system up to that point. For complexity, just note that $\mu(j+1) = \frac{j \cdot \mu(j) + d_{j+1}}{j+1}$, so the sequence of μ values, and their max, can be computed in linear time.

Now let the battery be bounded. Over the course of any region $[i, j]$, the best that can be hoped for is that the peak request will be reduced to $B/(j - i + 1)$ less than the average d_i in the region, i.e., $\rho(i, j)$, so no threshold lower than Algorithm *1.b*'s is possible.

For feasibility, it suffices to show that the battery level will be nonnegative after each time j. Suppose j is the first time underflow occurs. Let $i - 1$ be the last timeslot prior

to j with a full battery. Then there is no underflow *or* overflow in $[i, j)$, and so for each $t \in [i, j]$ the discharge at t is $b_t - b_{t+1} = d_t - T$ (possibly negative, meaning a charge) and the so the total net discharge over $[i, j]$ is $\sum_{t=i}^{j} d_t - (j - i + 1)T$. Total net discharge greater than B implies $T < \frac{-B+\sum_{t=i}^{j} d_t}{j-i+1}$, which contradicts the definition of T. The densest region can be found in $O(n^2)$, with n separate linear-time passes, each of which finds the densest region beginning in some position i, since $\rho(i, j + 1)$ can be computed in constant time from $\rho(i, j)$. □

3.3.1 Battery Level Boundary Conditions

We assumed above that the battery starts empty for the unbounded offline algorithm and starts full for the bounded offline algorithm, with the final battery level left indeterminate for both settings. A more general offline problem may require that $b_1 = \beta_1$ and $b_{n+1} = \beta_2$, i.e., the battery begins and ends at some charge levels specified by parameters β_1 and β_2. We argue here that these requirements are not significant algorithmically, since by pre- and postprocessing, we can reduce to the default cases for both the unbounded and bounded versions.

First, consider the unbounded setting, in which the initial battery level is 0. In order to enforce that $b_1 = \beta_1$ and $b_{n+1} = \beta_2$, run the usual optimal algorithm on the sequence $(d_1 - \beta_1, d_2, ..., d_{n-1}, d_n + \beta_2)$. (Recall that negative demands are allowed.) Then b_{n+1} will be at least β_2 larger than d_n. To correct for any surplus, manually delete a total of $b_{n+1} - \beta_2$ from the final requests. For the bounded setting, the default case is $b_1 = B$ and b_{n+1} indeterminate. To support $b_1 = \beta_1 \neq B$ and $b_{n+1} = \beta_2$, modify the demand sequence as above, except with $d_1 + (B - \beta_1)$ as the first demand and then do similar postprocessing as in the unbounded case to deal with any final surplus.

3.3.2 Optimal Battery Size

A large component of the fixed initial cost of the system will depend on battery capacity. A related problem therefore is finding the optimal battery size B for a given demand curve d_i, given that the battery starts and ends at the same level β (which can be seen as an amount *borrowed and repaid*). The optimal peak request possible will be $\frac{1}{n}\sum_{i=1}^{n} d_i = \mu(n)$, and the goal is to find the smallest B and β that achieve peak $\mu(n)$. (A completely flat request curve is possible given a sufficiently large battery.) This can be done in $O(n)$.

Since we will have $b_1 = b_{n+1}$, $r_i = \mu$ for all i, and $\sum d_i = \sum r_i$, there must be no overflow. Let $d'_i = d_i - \mu$, i.e., the amount by which d_i is above average (positive means discharge, negative means charge). Then the minimum possible $\beta = b_1$ is the maximum prefix sum of the d' curve (which will be at least 0). It could happen that the battery level will at some point rise above b_1, however. (Consider the example $d = (0, 0, 1, 0, 0, 0)$, for which $\mu = 1/6$, $d' = (-1/6, -1/6, 5/6, -1/6, -1/6, -1/6)$ and $\beta = 1/2$.) The needed capacity B can be computed as β plus the maximum prefix sum of *the negation of* the d' curve (which will also be at least 0). (In the example, we have $B = \beta + 2/6 = 5/6$.)

Although B is computed using β, we emphasize that the computed β is the minimum possible *regardless of* B, and the computed B is the minimum possible *regardless of* β.

3.4 Online Problem

We consider two natural choices of objective function for the online problem. One option is to compare the peak requests, so that if ALG is the peak request of the online algorithm ALG and OPT is that of the optimal offline algorithm OPT, then a c-competitive algorithm for $c \geq 1$ must satisfy $\frac{ALG}{OPT} \leq c$ for every demand sequence. Although this may be the most

natural candidate, we argue that for many settings it is uninteresting. If the peak demand is a factor k larger than the battery capacity, for example, then the trivial online algorithm that *does not use* the battery would be $(k/(k-1))$-competitive. If we drop the assumption of a small battery, then no reasonable competitive factor is possible in general, *even if D is revealed in advance*.

Proposition 3.4.1. *With peak demand D* revealed in advance *and finite time horizon n, no online algorithm* for the problem of minimizing peak request *can have competitive ratio 1) better than n if the battery begins with some strictly positive charge, or 2) better than $\Omega(\sqrt{n})$ if the battery begins empty.*[1]

Proof. For part 1, suppose that the battery begins with initial charge D. Consider the demand curve $(D, 0, ..., 0, ?)$, where the last demand is either D or 0. ALG *must* discharge D at time 1, since $OPT = 0$ when $d_n = 0$. Thus ALG's battery is empty at time 2. If ALG requests nothing between times 2 and $n-1$, and $d_n = D$, then we have $OPT = D/n$ and $ALG = D$; if ALG requests some $\alpha > 0$ during any of those timeslots, and $d_n = 0$, then we have $OPT = 0$ and $ALG = \alpha$. This yields a lower bound of n.

For part 2, suppose the battery begins empty, which is a disadvantage for both ALG and OPT. Consider the demand curve $(0, 0, ..., 0, D)$, in which case $OPT = D/n$. If an algorithm is c-competitive for some $c \geq 1$, then in each of the first $n-1$ timeslots of this demand curve ALG can charge at most amount cD/n. Now suppose that the only nonzero demand, of value D, arrives possibly earlier, at some timeslot $k \in [1, n]$, following $k-1$ demands of zero, during which ALG can charge at most $(k-1)cD/n$. In this case, we have $OPT = D/k$ and $ALG \geq D - (k-1)cD/n$, which yields the competitive ratio:

$$c \geq \frac{D - (k-1)cD/n}{D/k} = \frac{1 - (k-1)c/n}{1/k} = k - k^2c/n + kc/n$$

[1] *If the peak is not known, a lower bound of n can be obtained also for the latter case.*

Solving for c, and then choosing $k = \sqrt{n}$, we have:

$$c \geq \frac{k}{1 + k^2/n - k/n} = \frac{\sqrt{n}}{1 + 1 - 1/\sqrt{n}} = \Omega(\sqrt{n})$$

thus establishing the lower bound. □

Instead, we compare the *peak shaving amount* (or *savings*), i.e., $D - R$. For a given input, let *OPT* be the peak shaving of the optimal algorithm, and let *ALG* be the peak shaving of the online algorithm. Then an online algorithm is c-competitive for $c \geq 1$ if $c \geq \frac{OPT}{ALG}$ for every problem instance. For this setting, we obtain the online algorithms described below.

The online algorithms (see Def. 3.4.1) are shown in Table 3.2.

Alg.	battery	threshold T_i	per-slot time
2.a	both	$D - \frac{D - T_i^{opt}}{H_n}$	$O(n)$
2.b	both	$D - \frac{D - \rho(s_i, i)}{H_{n-s_i+1}}$	$O(1)$

Table 3.2: Threshold functions used for online algorithms.

Definition 3.4.1. *Let T_i^{opt} be the optimal threshold used by the appropriate optimal algorithm when run on the first i timeslots. At time i during the online computation, let s_i be the index of the most recent time prior to i with $b_{s_i} = B$ (or 1 in the unbounded setting).*

3.4.1 Lower Bounds for $D - R$

Since the competitiveness of the online algorithms holds for arbitrary initial battery level, in obtaining lower bounds on competitiveness, we assume particular initial battery levels.

Proposition 3.4.2. *With peak demand D unknown and finite time horizon n, there is no online algorithm 1) with any constant competitive ratio for unbounded battery (even with $n = 2$) or 2) with competitive ratio better than n for bounded battery.*

Proof. For part 1, assume $b_1 = 0$, and suppose $d_1 = 0$. Then if ALG requests $r_1 = 0$ and we have $d_2 = D$, then $OPT = D/2$ and $ALG = 0$; if ALG requests $r_1 = a$ (for some $a > 0$) and we have $d_2 = a$, then $OPT = a/2$ and $ALG = 0$. For part 2, let $b_1 = B$, and assume ALG is c-competitive. Consider the demand curve $(B, 0, 0, \ldots, 0)$. Then OPT clearly discharges B at time 1 (decreasing the peak by B). For ALG to be c-competitive, it must discharge at least $\frac{B}{c}$ in the first slot. Now consider curve $(B, 2B, 0, 0, \ldots, 0)$. At time 2, OPT discharges B, decreasing the peak by B, so at time 2, ALG must discharge at least $\frac{B}{c}$. (At time 1, ALG already had to discharge $\frac{B}{c}$.) Similarly, at time i for $(B, 2B, 3B, ..., iB, ..., nB)$, ALG must discharge $\frac{B}{c}$. Total discharging by ALG is then at least: $\sum_{i=1}^{n} \frac{B}{c} = \frac{nB}{c}$. Since we must have $n\frac{B}{c} \leq B$, it follows that $c \geq n$. □

The trivial algorithm that simply discharges amount B/n at each of the n timeslots and *never charges* is n-competitive (since $OPT \leq B$) and so matches the lower bound for the bounded case.

Proposition 3.4.3. *With peak demand D known in advance and finite time horizon n, no online algorithm can have 1) competitive ratio better than H_n if the battery begins nonempty or 2) competitive ratio better than $H_n - 1/2$ if the battery begins empty, regardless of whether the battery is bounded or not.*

Proof. First assume the battery has initial charge b. (The capacity is either at least b or unbounded.) Suppose ALG is c-competitive. Consider the curve $(D, 0, 0, \ldots, 0)$, with $D \geq b$. Then OPT clearly discharges b at time 1 (decreasing the peak by b). For ALG to be c-competitive, it must discharge at least $\frac{b}{c}$. Now consider curve $(D, D, 0, 0, \ldots, 0)$. At times 1 and 2, OPT discharges $\frac{b}{2}$, decreasing the peak by $\frac{b}{2}$. At time 2, ALG will have to discharge at least $\frac{b/2}{c} = \frac{b}{2c}$. Similarly, at time i on $(D, D, D, ..., D)$, ALG must discharge

$\frac{b}{ic}$. Total discharging by ALG is then at least: $\sum_{i=1}^{n} \frac{b}{ic} = H_n \frac{b}{c}$. Since we discharge at each timeslot and never charge, we must have $\frac{b}{c} H_n \leq b$, and so it follows that $c \geq H_n$.

Now let the battery start empty. Assume the battery capacity is at least D or is unbounded. Repeat the argument as above, except now with a zero demand inserted at the start of the demand curves, which gives both ALG and OPT an opportunity to charge. Then for each time $i \in [2, n]$, ALG must discharge at least $\frac{D}{ic}$ since OPT may discharge (and so save) $\frac{D}{i}$ (in which case it would have initially charged $D(1 - 1/i)$). ALG is then required to discharge $(H_n - 1)\frac{D}{c}$ during the last $n-1$ timeslots. Obviously it could not have charged more than D during the first timeslot. In fact, it must charge less than this. On the sequence $(0, D, 0, 0, ..., 0)$, OPT charges $D/2$ at time 1 and discharges it at time 2, saving $D/2$. ALG must discharge $\frac{D}{2c}$ at time 2 in order to be c-competitive on this sequence, and so reduce the peak D by $\frac{D}{2c}$. Therefore at time 1, ALG cannot charge more than $D - \frac{D}{2c}$. Therefore we must have $D - \frac{D}{2c} \geq (H_n - 1)D/c$, which implies that $c \geq H_n - 1/2$. □

3.4.2 Bounded Battery

Our first online algorithm bases its threshold at time i on a computation of the optimal offline threshold T_i^{opt} for the demands $d_1, ..., d_i$. The second bases its threshold at time i on $\rho(s_i, i)$ (see Defs. 3.3 and 3.4.1). Assuming the algorithms are feasible (i.e., no battery underflow occurs), it is not difficult to show that they are competitive.

Theorem 3.4.4. *Algorithms* 2.a *and* 2.b *are H_n-competitive, if they are feasible, and have* per-timeslot *running times of $O(n)$ and $O(1)$, respectively.*

Proof. First observe that $\hat{\rho}(i) \geq \rho(s_i, i)$ implies $\frac{D - \hat{\rho}(i)}{H_n} \leq \frac{D - \rho(s_i, i)}{H_{n - s_i + 1}}$ implies $T_i^a \geq T_i^b$ for all i. Therefore it suffices to prove competitiveness for Algorithm *2.a*. Since T_i^{opt} is the lowest possible threshold up to time i, $D - T_i^{opt}$ is the highest possible peak shaving as of

time i. Since the algorithm always saves a $1/H_n$ fraction of this, it is H_n-competitive by construction.

Since $\mu(1, i+1)$ can be found in constant time from $\mu(1, i)$, Algorithm *2.b* is constant-time per-slot. Similarly, Algorithm *2.a* is, recalling the proof of Theorem 3.3.1, linear per-slot.

We now show that indeed both algorithms are feasible, using the following lemma, which allows us to limit our attention to a certain family of demand sequences. □

Lemma 3.4.5. *If there is a demand sequence* $(d_1, d_2, ..., d_n)$ *in which underflow occurs for Algorithm* 2.a *or* 2.b*, then there is also a demand sequence (for the same algorithm) in which underflow continues to the end (i.e.,* $b_{n+1} < 0$*) and no overflow ever occurs, i.e., one in which the battery level decreases monotonically from full to empty.*

Proof. The battery is initialized to full, $b_1 = B$. Over the course of running one of the algorithms on a particular problem instance, the battery level will fall and rise, and may return to full charge multiple times. Suppose underflow were to occur at some time t, i.e. $b_t < 0$, and let s be the most recent time before t when the battery was full. We now construct a demand sequence with the two desired properties, for both algorithms.

First, if $s > 1$, then also considering region $[1, s-1]$ when defining the threshold T_i for Algorithm *2.a* or *2.b* can only raise the threshold over what it would be if only region $[s, t]$ were considered. Therefore shifting the region leftward from $[s, t]$ to $[1, t'] = [1, t-s+1]$ will only lower the thresholds used, which therefore preserves the underflow. Second, since any underflow that occurs in region $[1, t']$ can be extended to the end of sequence by setting each demand after time t' to D, we can assume wlog that $t' = n$. □

Theorem 3.4.6. *Algorithms* 2.a *and* 2.b *are feasible.*

Proof. For a proof by contradiction, we can restrict ourselves by Lemma 1 to regions that begin with a full battery, underflow at the end, and have no overflow in the middle. For such a region, the change in battery-level is well behaved ($b_i - b_{i+1} = d_i - T_i$), which allows us to sum the net discharge and prove it is bounded by B. We now show that it is impossible for the battery to fall below 0 at time n, by upperbounding the *net discharge* over this region. Let $\Delta b_i = b_i - b_{i+1} = d_i - T_i$ be the amount of energy the battery discharges at step i. (Δb_i will be negative when the battery charges.) We will show that $\sum_{1 \le i \le n} \Delta b_i \le B$. Let Δb_i^a and Δb_i^b refer to the change in battery levels for the corresponding algorithms. Because as we observed above $T_i^a \ge T_i^b$, we have:

$$\Delta b_i^a = d_i - T_i^a \le \Delta b_i^b = d_i - T_i^b$$

Therefore it suffices to prove the feasibility result for Algorithm *2.b*, and so we drop the superscripts. Expanding the definition of that algorithm's threshold, we have:

$$\Delta b_i = d_i - T_i = d_i - \left(D - \frac{1}{H_n}(D - \rho(1,i)) \right) = d_i - \left(D - \frac{1}{H_n}\Big(D - \frac{1}{i}\Big(\sum_{k=1}^{i} d_k - B \Big)\Big)\right) \tag{3.4.1}$$

By summing Eq. 1 for each i, we obtain:

$$\begin{aligned}
\sum_{i=1}^{n} \Delta b_i &= \sum_{i=1}^{n} \left(d_i - \Big(D - \frac{D - (\sum_{k=1}^{i} d_k - B)/i}{H_n} \Big) \right) \\
&= \sum_{i=1}^{n} \left(d_i - \Big(D - \frac{D - (\sum_{k=1}^{i} d_k)/i}{H_n} \Big) \right) + \sum_{i=1}^{n} \frac{B/i}{H_n} \\
&= \sum_{i=1}^{n} \left(d_i - \Big(D - \frac{D - (\sum_{k=1}^{i} d_k)/i}{H_n} \Big) \right) + B
\end{aligned}$$

Therefore it suffices to show that:

$$\sum_{i=1}^{n} \Big(d_i - \big(D - \frac{D - \sum_{k=1}^{i} d_k/i}{H_n} \big) \Big) \le 0 \tag{3.4.2}$$

which is equivalent to:

$$\sum_{i=1}^{n} d_i - nD + \frac{1}{H_n}\Big(nD - \sum_{i=1}^{n}\sum_{k=1}^{i} d_k/i\Big) \leq 0$$

$$\textit{iff } \sum_{i=1}^{n} d_i + nD(\frac{1}{H_n} - 1) - \frac{1}{H_n}\Big(\sum_{i=1}^{n}\sum_{k=1}^{i} d_k/i\Big) \leq 0$$

$$\textit{iff } \sum_{i=1}^{n} H_n d_i - \Big(\sum_{i=1}^{n}\sum_{k=1}^{i} \frac{d_k}{i}\Big) \leq nD(H_n - 1) \tag{3.4.3}$$

With the following derivation:

$$\sum_{i=1}^{n}\sum_{k=1}^{i}\frac{d_k}{i} = \sum_{i=1}^{n}\sum_{k=1}^{n}\frac{d_k}{i} - \sum_{i=1}^{n}\sum_{k=i+1}^{n}\frac{d_k}{i} = \sum_{k=1}^{n}\sum_{i=1}^{n}\frac{d_k}{i} - \sum_{k=1}^{n}\sum_{i=1}^{k-1}\frac{d_k}{i} = \sum_{k=1}^{n} H_n d_k - \sum_{k=1}^{n} H_{k-1} d_k$$

we can rewrite Eq. 3 (replacing the parenthesized expression) as: $\sum_{i=1}^{n} H_{i-1} d_i \leq n(H_n - 1)D$. Since $d_i \leq D$ and $D > 0$, it suffices to show that: $\sum_{i=1}^{n} H_{i-1} \leq n(H_n - 1)$. In fact, this holds with equality (see [49], Eq. 2.36). □

3.4.3 Unbounded Battery and Boundary Conditions

Both online algorithms, modified to call appropriate subroutines, also work for the unbounded battery setting. The algorithms are feasible in this setting since $\rho(i) \leq T_i^{opt}$ still holds, where T_i^{opt} is now the optimal threshold for the unbounded battery setting. (Recall that offline Algorithm *1.a* can "greedily" run in linear total time.) The algorithm is H_n-competitive by construction, as before.

Corollary 3.4.7. *Algorithms* 2.a *and* 2.b *are feasible in the unbounded battery setting.*

Proof. The proof is similar to that of Theorem 3.4.6, except that b_1 (which may be 0) is plugged in for all occurrences of B (resulting in a modified ρ), and overflow is no longer a concern. □

We also note that the correctness and competitiveness of both algorithms (with minor preprocessing) holds for the setting of arbitrary initial battery levels. In the case of Algorithm *2.a*, each computation of T_i^{opt} is computed for the chosen b_1, as described in Section 3.3.1, and the online algorithm again provides a fraction $1/H_n$ of this savings. Although in the bounded setting "increasing" b_1 for the T_i^{opt} computation by modifying d_1 may raise the peak demand *in the offline algorithm's input*, the D value for the online algorithm is not changed. Indeed, examining the proof of Theorem 3.4.6, we note that the upperbound on the d_i is only used for $i > 1$ (see Eq. 4).

3.5 Conclusion

In this chapter, we formulated a novel peak-shaving problem, and gave efficient optimal offline algorithms and optimally competitive online algorithms. In work in progress, we are testing our online algorithms on actual client data from Gaia [1]. (See [14] for preliminary results.) There are several interesting extensions to the theoretical problem that we plan to address, such as adapting the algorithms to inefficient batteries that lose a percentage of charge instantly or over time or batteries with a charging speed limit. We could also optimize for a moving average of demands rather than a single peak. Finally, online algorithms could also be granted additional predictions about the future.

Appendix: Solving the offline problem by general techniques

As mentioned above, the offline problem can be solved by general techniques, such as LP or (generalized) parametric flow [7]. While we include these approaches for completeness, our offline algorithms are significantly faster and simpler, and easily extend to the online problem.

A linear program formalization of the offline problem is shown below. In the LP, request r_i is divided into the portion $curr_i$ used immediately and the portion ch_i charged to the battery; demand d_i is divided into portion $curr_i$ obtained from the current request and the portion dis_i discharged from the battery. Constant ℓ is the coefficients for energy loss; constants β_0 and β_n indicate the starting and required ending battery levels. This formulation can easily be generalized to some of the future extensions we suggest in the conclusion.

Minimize: R

Such that: $R \geq r_i$, for each i,

$r_i = curr_i + ch_i$, for each i,

$d_i = curr_i + dis_i$, for each i,

$b_{i+1} = b_i - dis_i + \ell ch_i$, for each i

$b_i \leq B$, for each i,

$b_0 = \beta_0$ and $b_n = \beta_n$,

$r_i, curr_i, ch_i, dis_i, b_i \geq 0$, for each i

Another way of solving the offline problem is via (generalized) parametric max-flow. Solving max-flow on the network flow drawing below will determine whether all demands can be satisfied with requests at most P, where P is the (parameter) edge capacity on edges

between the source nodes and the demand nodes. This model allows for battery capacity, time loss (coefficient c_1) and entry loss (coefficient c_2), and required initial and final battery levels.

Chapter 4

Lossy Batteries

4.1 Introduction

In this chapter, extend the battery charge problem to more realistic batteries. The battery (previously assumed to be perfectly efficient) is used to store energy for later use. We now extend the model to the more realistic setting of *lossy* batteries, which lose to conversion inefficiency a constant fraction of any amount charged (e.g. 33%). We will consider batteries with and without a bounded capacity, and with or without a percentage loss in charging due to inefficiency. In the online version of this problem, the algorithm must determine each request without knowledge of future demands or free source availability, with the goal of maximizing the amount by which the peak is reduced.

We will give efficient optimal algorithms for the offline problem, with and without a bounded battery. Central to our analysis is a mathematical property we call a *generalized average*. Our fastest offline algorithms for the lossy battery settings compute a series of generalized averages with the aid of balanced binary search trees. We also show how to find the optimal offline battery size, given the requirement that the final battery level equals

the initial battery level. Finally, we will give efficient H_n-competitive algorithms assuming the peak *effective* demand is revealed in advance, and provide matching lower bounds.

4.2 Model and Algorithms

The problem model and objective are the same as in the lossless setting (see Section 3.2), except for the introduction of battery loss. The loss model works as follows. For each unit of energy charged, only $r = 1 - \ell$ units will be available for discharge, due to the combined inefficiencies of charging and discharging. This loss could be broken into separate components ℓ_1, ℓ_2 for charge and discharge, but since the loss does not depend on time, doing so would have essentially no effect. For simplicity, we merge these losses into a single loss that occurs instantly at the time of charging.

Threshold algorithms. For a particular snapshot (d_i, r_i, b_i), demand d_i must be supplied through a combination of the request r_i and a change in battery $b_i - b_{i-1}$. This means that there are only three possible modes for each timestep: request exactly the demand, request more than the demand and *charge* the difference, or request less than the demand and *discharge* the difference. Our online and offline algorithms are *threshold algorithms*. If $(T_1, T_2, ..., T_n)$ are the chosen thresholds, then the algorithms (generalizing those given in Section 3.2) behave as follows:

```
for each timeslot i
    if d_i < T_i
        charge min(B − b_i, T_i − d_i)
    else
        discharge min(d_i − T_i, b_i)
        if d_i − T_i < b_i
            T_i ← T_i + (d_i − T_i − b_i)
```

The algorithm schema amounts to the rule: *at each timeslot i, request an amount as near to T_i as the battery constraints will allow*. Our offline algorithms use a constant T (though in practice an offline algorithm could naturally lower its requests to avoid overflow); our online algorithms compute T_i dynamically for each timeslot i.

Definition 4.2.1. *Let* overflow *be the situation in which $T_i - d_i > B - b_i$, i.e., there is not enough room in the battery for the amount we want to charge. Let* underflow *be the situation in which $d_i - T_i > b_i$, i.e., there is not enough energy in the battery for the amount we want to discharge. Call a threshold algorithm* feasible *if underflow never occurs (overflow merely indicates a lower effective request).*

The second *if* statement of the algorithm schemas is executed only if underflow occurs. The competitiveness guarantee of Algorithm *2.b* for the lossless setting was achieved in [15] by showing that such underflow would never occur. The factor-revealing LP below provides evidence that such underflow also never occurs in the lossy setting. Our heuristic algorithms choose lower, more aggressive thresholds, with the result that such underflow does (or rather would) occur. Since meeting demand is a strict requirement, in the event of underflow, the request rises accordingly to keep the battery level non-negative, which is what the *if* statement does.

4.3 GA and Lossy Battery Algorithms

The algorithms for lossy batteries are structurally similar to those for lossless, except that computations of *average* are replaced with *generalized average* (GA). In all cases, the average computed over an interval will correspond to the best possible maximum request over that interval, which can be found by examining all subintervals. The algorithms used

are shown below:

Alg.	online	threshold T_i	running-time
1	no	$\hat{\mu}(1,n)$	$O(n^2 \log n)$
2.a	yes	$D - \frac{D-\hat{\mu}(1,i)}{H_n}$	$O(n^2 \log n)$
2.b	yes	$D - \frac{D-\mu(s_i,i)}{H_{(n-s_i+1)}}$	$O(n \log n)$

Definition 4.3.1. *Given n real values $(y_1, y_2, \ldots, y_n)$ and constants $0 < r \leq 1$ and $B \geq 0$, let the **generalized average** $GA(y_1, y_2, \ldots, y_n)$ be the value μ satisfying $U(a) = B + r \cdot L(a)$, where: $U(a) = \sum_{i=1}^{n} \max(y_i - a, 0)$ and $L(a) = \sum_{i=1}^{n} \max(a - y_i, 0)$. We call $U(a)$ and $L(a)$ μ's L/U* values *or μ's* upper *and* lower. *Treating the input values $y_1, ..., y_n$ as a step function $y = y(x)$, they correspond to the area above μ and below y (upper) and the area below μ and above y (lower).*

Some intuition may be provided by considering what is likely the simplest way, in terms of coding, of computing a GA: binary search in the range $[-B, \max_i\{d_i\}]$. For each candidate value μ, if $B + r \cdot L(\mu) > U(\mu)$ then shift downward and otherwise shift upward, until the two values are sufficiently close.

Note that μ need not be one of the y_i values. When $B = 0$ and $r = 1$, the generalized average is simply the mean of the values y_i; when $B = 0$ and r approaches 0, the generalized average approaches the maximum.

Computing a GA in $O(n \log n)$ is not difficult. First sort the values y_i, and let the values' subscripts now reflect their new positions. Next, set $L(y_1) = 0$, since y_1 is the smallest y_i. For each subsequent i up to n, $L(y_i)$ can be computed in constant time (we omit details due to space constraints). Similarly, compute each $U(y_i)$, starting with $U(y_n)$. Once all the U/Ls are computed, μ's *neighbors* (y_i, y_{i+1}), i.e., the two nearest input values that μ

```
GenAvgNbrs(A[], X_U, X_L, W_U, W_L) :
    if length(A) == 2
        return A;
    else  p = Select-Median(A);                          (a)
        (A_L, A_U) = Pivot(A,p);                         (b)
        U_p = Upper(A,p); L_p = Lower(A,p);              (c)
        U = U_p + X_U + W_U · (max(A) − p); L = L_p + X_L + W_L · (p − min(A));
        if U < r · L + B
            return GenAvgNbrs(A_U ∪ p, L, X_U, W_L + |A_L|, W_U);
        else if U > r · L + B
            return GenAvgNbrs(A_L ∪ p, X_L, U, W_L, W_U + |A_U|);
        else
            return p;
```

lies between, can be found by inspection, and given these μ can be computed in constant time. Unlike the ordinary arithmetic mean, however, computing a GA in $O(n)$ requires more effort.

Our recursive algorithm, whose behavior we sketch in words, both is inspired by the well-known linear-time deterministic Selection Algorithm [34], and calls it as a subroutine. The bulk of the work is in finding μ's *neighbors*. Given these data points (and their L/U values), we can solve for the correct value μ in constant time. (The cases–*not shown in the pseudocode*–when the solution μ is among the data points, and when μ is less than *all* the points can be checked as special cases.) The algorithm for finding the neighboring data points to μ takes the set of points y_i as input. Let $0 \leq r < 1$ and B be the parameters to the GA. The first parameter to the algorithm is the set of values to be averaged; all other parameters to the first (non-recursive) call are set to 0.

Theorem 4.3.1. *$GA(y_1, ..., y_n)$ can be computed in $O(n)$ time.*

Proof. (sketch) With $|A| = n$, lines a,b,c each take time $O(n)$ since *Select-Median* uses the Selection Algorithm, *Pivot* is the usual Quicksort pivoting algorithm, and Upper and

Lower are computed directly. (Min and max can be passed in separately, but we omit them for simplicity.) The function makes one recursive call, whose input size is by construction half the original input size. Hence the total running time is $O(n)$. □

The bulk of the work done by our algorithms for lossy batteries is to compute the GA for a series of ranges $[i, j]$, as i stays fixed (as e.g. 1) and j increases iteratively (e.g. from 1 to n). It is straightforward to do this in $O(n^2)$ time, by maintaining a sorted sublist of the previous elements, inserting each new y_j and computing the new GA in linear time. Unlike ordinary averages, $GA[i, j]$ and the value y_{j+1} do not together determine $GA[i, j + 1]$.[1] (The GA could also be computed separately for each region $[1, j]$.) This yields offline algorithms for the lossy unbounded and bounded settings, with running times $O(n^2)$ and $O(n^3)$. Through careful use of data structures, we obtain faster algorithms, with running times $O(n \log n)$ and $O(n^2 \log n)$, respectively.

Theorem 4.3.2. *The values $GA[1, j]$, as j ranges from 1 to n can be computed in $O(n \log n)$.*

Proof. (sketch) A balanced BST is used to store previous work so that going from $GA[i, j]$ to $GA[i, j + 1]$ is done in $O(\log n)$. Each tree node stores a y_i value plus other data (its L/U, etc.) used by GENAVGNBRS to run in $O(\log n)$. Each time a new data point y_i is inserted into the tree, its data must be computed (and the tree must be rebalanced). Unfortunately, each insertion *partly* corrupts *all other nodes' data*. Using a lazy evaluation strategy, we initially update only $O(\log n)$ values. After the insert, GENAVGNBRS is run on the tree's current set of data points, in $O(\log n)$ time, relying only on the nodes' data that is guaranteed to be correct. Running on the BST, GENAVGNBRS's subroutines (Select-Median, Pivot, and selection of the subset to recurse on) now complete in $O(\log n)$, for a

[1] For example, when $B = 10$ and $r = .5$, $GA(5, 10, 15) = GA(3, 21, 3) = 7$, but $GA(5, 10, 15, 20) = 10.83 \neq GA(3, 21, 3, 20) = 11.33$.

total of $O(n \log n)$. □

Definition 4.3.2. *Let $\mu(i,j)$ be the GA of the demands over region $[i,j]$. Let $\hat{\mu}(h,k) = \max_{h \le i \le j \le k} \mu(i,j)$. At time i, let s_i be the most recent time when the battery was full.*

Theorem 4.3.3. *For the offline/lossy setting, Algorithm 1 ($T_i = \hat{\mu}(1,n)$) is optimal, feasible, and runs in time $O(n^2 \log n)$.*

Proof. Within any region $[i,j]$, the battery may help in two ways. First, the battery may be able to lower the local peak by sometimes charging and sometimes discharging. Second, the battery in the best case would start with charge B at timestep i. With battery loss percentage ℓ, the total amount discharged from the battery over this period can be at most B plus $(1-\ell)$ times the total amount charged. The optimal threshold over this region cannot be less than $GA(d_i, ..., d_j)$ with $(1-\ell, B)$ chosen as its parameters (r, B).

The threshold used is $T = \hat{\mu}(1,n)$. It suffices to show that the battery will be non-negative after each time j. Suppose j is the first time underflow occurs. Let $i-1$ be the last timestep prior j with a full battery (or 0 if this has never occurred). Then there is no underflow *or* overflow in $[i,j)$, so the total charged in region $[i,j]$ is exactly $U(T) = \sum_{t=i}^{j} \max(T - d_t, 0)$ and the total discharged will be $L(T) = \sum_{t=i}^{j} \max(d_t - T, 0)$. The amount of energy available for discharge over the entire period is $B + r \cdot L(T)$. Overflow at time j means $U(T) > B + r \cdot L(T)$, but this contradicts the definition of T.

To compute the thresholds, compute $GA[i,j]$ iteratively (varying j from i to n) for each value i. Each i value takes $O(n \log n)$, for a total of $O(n^2 \log n)$. □

Corollary 4.3.4. *If the battery is effectively unbounded, then a similar optimal algorithm can be obtained, which runs in time $O(n \log n)$.*

4.4 Factor-revealing LPs

If no underflow occurs, then algorithms *2.a* and *2.b* are H_n-competitive by construction. (Recall that the objective function is the peak reduction amount.) In this section, we use the factor-revealing LP technique of Jain et al. [63] to provide some quasi-empirical evidence that no such underflow can ever occur.

A factor-revealing LP is defined based on a particular algorithm for a problem. The LP variables correspond to possible instances, of a certain size n, of the optimization problem. (We therefore have an indexed family of linear programs.) The optimal solution value of the linear program reveals something about the algorithm it is based on. In the original Facility Location application, the objective function was the ratio of the cost incurred by the approximation algorithm in covering the facility and the optimal cost (assumed wlog to be 1) of doing so, so the maximum possible value of this ratio provided an upper bound on the algorithm's approximation guarantee.

The size index of our LPs is the number of timesteps n. The objective function is the final battery level b_{n+1}. The constraints are properties describing the behavior of the algorithm; some of the constraints perform book-keeping, including keeping track of the battery level over time. We first provide the factor-revealing LP for the lossless setting (Fig. 4.1), which is simpler than the lossy.

We now explain this program. The battery is initialized to B and can never supersede this level. As we argue below, we can limit ourselves without loss of generality to demand sequences in which the algorithm never wishes to charge to a level greater than B, i.e. no overflow occurs. For such inputs, the threshold scheme's first min has no effect and we always have that $b_{i+1} = b_i + T_i - d_i$. Threshold T_i is constrained in the LP to equal the expression for Algorithm *2.b*'s threshold, with opt_i lower-bounded by the closed-form

min: b_{n+1}
s.t.: $b_{i+1} = b_i + T_i - d_i$, for all i
$b_i \leq B$
$T_i = D - (D - opt_i)/H_n$, for all i
$opt_i \geq (1/i)(-B + \sum_{j=1}^{i} d_j)$, for all i

$b_1 = B$
$d_i \leq D$, for all i
$D \geq 0, B = 1$

Figure 4.1: Factor-revealing linear program for lossless batteries (LP1)

expression for the analog of GA for lossless batteries [15]. Moreover, this value is less than or equal to the corresponding value used in Algorithm *2.a*, with the effect that T_i is less than or equal to the corresponding threshold of Algorithm *2.a at every time* i. This in turn means that feasibility of Algorithm *2.b* implies feasibility of Algorithm *2.a*. D is included in the program for clarity.

We solved this LP, written in AMLP, with LP solvers on the NEOS server [2], for several values $n \leq 100$. The solution value found was 0, consistent with the known result (i.e., Theorem 3.4.6 above). Lemma 3.4.5 above allows us to limit our attention to a certain family of demand sequences, viz., those in which once underflow occurs it continues to the end (i.e., $b_n < 0$) and no overflow ever occurs.

Theorem 4.4.1. *If the optimal solution value LP1, for parameter size* n*, is at least 0, then Algorithms* 2.a *is* H_n*-competitive in the lossless setting, for problem size* n*.*

Proof. Suppose underflow were to occur at some time t, and let s be the most recent time prior to t when the battery was full. Then by the lemma, $[s, t]$ can be assumed wlog to be $[1, n]$. The assumptions that the battery starts at level B and never reaches this level again (though it may rise and fall non-monotonically) are implemented by the constraints

stating that $b_1 = B$ and $b_i \geq B$. Since no overflow occurs, the first min in the threshold algorithm definition has no effect, and the battery level changes based only on T_i and d_i, i.e., it rises by amount $T_i - d_i$ (which may be negative), which is stated in constraints. Since $opt_i = \hat{\mu}(1, i)$ is the max of expressions for all sequences of $[1, i]$, in particular we have that $opt_i \geq (-B + \sum_{j=1}^{i} d_i)/i$. The optimal solution value to such an LP equals the lowest possible battery level which can occur, given any possible problem instance of size n, when using any algorithm consistent the these constraints. Since in particular the behavior of Algorithms *2.a* is consistent with these constraints, the result follows. □

Corollary 4.4.2. *If the optimal solution value LP1, for all parameter sizes $\leq n$, is at least 0, then Algorithm* 2.b *is H_n-competitive in the lossless setting, for problem sizes $\leq n$.*

Proof. In Algorithm *2.b*, threshold T_i is defined based on the region beginning after the last overflow at position $s = s_i$, shifting $[s, t]$ to $[1, t'] = [1, t - s + 1]$ has *no effect*. If *2.b instead* always used H_n, then the lemma above would directly apply, since the algorithm would then perform identically on $[s, t]$ to $[1, t']$, and then the underflow could again be extended to the end. The result that LP1's optimal solution value is ≥ 0 would then imply that this *modified* Algorithm *2.b* is feasible for inputs of size n.

In fact, the actual Algorithm *2.b* uses H_{n-s+1}, with $s = s_i$, at time i. In effect, Algorithm *2.b* treats the demand suffix $d_{s+1}, ..., d_n$ as an independent problem instance of size $n - s + 1$. Each time the battery overflows, the harmonic number subscript is modified, and there is a new subregion and a new possibility for underflow. If all sizes of overflow-free subregions are underflow-free, then the algorithm is feasible. Therefore, if LP1's optimal solution is non-negative, Algorithm *2.b* is feasible. □

The more complicated Factor-revealing program for the lossy setting is shown in Fig. 4.2. The additional difficulty here is that the program has quadratic constraints.

min: b_{n+1}
s.t.: $b_{i+1} = b_i + L \cdot ch_i - dis_i$, for all i
$ch_i \cdot dis_i = 0$ (*)
$b_i \leq B$, for all i
$ch_i, dis_i \geq 0$, for all i

$b_1 = B$
$D \geq d_i$, for all i
$B = 1, D \geq 0$

$T_i = D - (D - opt_i)/H_n$
$T_i = d_i - dis_i + ch_i$
$opt_i \geq ga_i$, for all i

$B + L \cdot (\sum_{j=1}^{i} cho_{i,j}) = \sum_{j=1}^{i} diso_{i,j}$, for all i (*)
$cho_{i,j} \cdot diso_{i,j} = 0$, for all $(j, i) : j \leq i$ (*)
$ga_i = d_j - diso_{i,j} + cho_{i,j}$, for all $(j, i) : j \leq i$
$cho_{i,j}, diso_{i,j} \geq 0$, for all $(j, i) : j \leq i$

Figure 4.2: Factor-revealing quadratically-constrained LP for lossy batteries (LP2)

We omit a full description of this program and only remark that the two main difficulties are 1) that there is an essential asymmetry in the battery behavior, which complicates the first constraints; and 2) that since we have no closed formula for GA, (a lower bound on) the optimal threshold must be *described* rather than *computed*.

There are three sets of quadratic constraints, indicated by stars. In fact, it is possible to remove these and convert LP2 to a linearly-constrained quadratic program, defined for a fixed constant efficiency L. Unfortunately, the resulting QP is not convex.

We then have the following results.

Theorem 4.4.3. *If the optimal solution value LP2, for parameter size n, is at least 0, then Algorithms* 2.a *is H_n-competitive in the lossy setting, for problem size n.*

Corollary 4.4.4. *If the optimal solution value LP2, for all parameter sizes $\leq n$, is at least*

0, then Algorithm 2.b *is H_n-competitive in the lossy setting, for problem sizes $\leq n$.*

We solved the second program, implemented in AMLP, using several solvers (MINLP, MINOS, SNOPT) on the NEOS server [2], for several values $n \leq 100$. (The number of variables is quadratic in n, and there are limits to the amount of memory NEOS provides.) In all case, we found the solution value found was (within a negligible distance of) non-negative. Although these solvers do not guarantee a globally optimal solution (at least not for non-convex quadratically-constrained LPs), we believe this performance provides some "quasi-empirical" evidence for the correctness of Algorithms *2.a* and *2.b*.

4.5 Conclusion

In this chapter, we presented optimal offline algorithms and both heuristic and possibly competitive online algorithms for the peak reduction problem with lossy batteries. The factor-revealing LPs for the lossy setting presently provide only quasi-empirical evidence for competitiveness. The potential future availability of global quadratically-constrained LP solvers, however, could provide computer-aided proof of such competitiveness, at least for instances of bounded size. Several additional future extensions suggest themselves:

- additional free but unreliable energy sources (e.g. solar power)
- limited battery charging/discharging speed
- battery loss over time ("self-discharge')
- multi-dimensional demands and resulting complex objective functions

Chapter 5

Sensor Assignment

5.1 Overview

Mission-centric sensor networks present many research challenges. One such challenge is how to best assign sensors to tasks, considering that there may be multiple tasks, of different priorities and information needs, running concurrently in the network, and sensors of multiple types available to meet those needs. Tasks may require one or more sensors, possibly of different types. Given this multiplicity of task types and needs, our goal is to assign specific sensors to the tasks in order to maximize the utility of the sensor network. This is especially challenging in environments that use *directional* sensors as each sensor in this case can be assigned to at most one task.

A sensor network deployed for monitoring applications may be tasked with achieving multiple, possibly conflicting, missions. Although certain types of sensors, such as seismic sensors, can receive data from their surroundings as a whole, other sensor types, such as cameras, are directional. In these cases, the direction of each sensor, and thus the mission it serves, must be chosen appropriately.

In many sensor network applications, it is necessary to support multiple missions that may arrive over time and compete for the same sensing resources. For *isotropic* sensing devices recording ambient temperature, for example, the information provided by a single sensor can be used to support any and all missions, given that they lie within its sensing range. A directional or *anisotropic* sensor, such as a camera, however, must be directed to a certain location and hence can in general only support a single mission. Thus *choices must be made* as to which sensors will be assigned to which missions. Given information about available sensors and missions, the network must have a process to choose the "best" assignment of sensors to missions.

An intelligent sensor network should direct its resources to the most important feasible missions, reallocating resources appropriately as missions arrive or depart, while taking care not to waste resources on unsuccessful missions. If there are multiple sensors and multiple missions, it must choose the best matching of sensors to missions. A given sensor may offer different missions varying amounts of information (because of geometry, obstructions, or remaining battery level, for example), or none at all.

5.2 Frugality

A second concern is frugality, i.e., the conservation of resources. We consider two broad classes of environments: static and dynamic. The *static* setting is motivated by situations in which different users are granted control over the sensor network at different times. During each time period, the current user may have many simultaneous missions. While the current user will want to satisfy as many of these as possible, sensing resources may be limited and expensive, both in terms of equipment and operational cost. In some environments, replacing batteries may be difficult, expensive, or dangerous. Furthermore, a sensor operating in

active mode (i.e., assigned to a mission) may be more visible than a dormant sensor, and so in greater danger of tampering or damage. Therefore, we give each mission in the static problem a budget so that no single user may overtax the network and deprive future users of resources. This budget serves as a constraint in terms of the amount of resources that can be allocated to a mission regardless of profit.

Our second environment is a *dynamic* setting in the sense that missions arrive over time and have different durations. In these cases explicit budgets may be too restrictive because we must react to new missions given our current operating environment, that is, the condition of the sensors will change over time. Instead, we use battery lifetime as a means of discouraging excessive assignment of sensors to any one mission, evaluating trade-offs between the relative value of a given assignment and the expected profit earned with each sensor (given the mission statistics).

This part presents results previously appearing in [12, 87, 65, 89].

Chapter 6

Assigning Sensors to Tasks

6.1 Introduction

In many sensor network applications, it is necessary to support multiple missions that may arrive over time and compete for the same sensing resources. For *isotropic* sensing devices recording ambient temperature, for example, the information provided by a single sensor can be used to support any and all missions, given that they lie within its sensing range. A directional or *anisotropic* sensor, such as a camera, however, must be directed to a certain location and hence can in general only support a single mission. Thus *choices must be made* as to which sensors will be assigned to which missions. Given information about available sensors and missions, the network must have a process to choose the "best" assignment of sensors to missions.

An intelligent sensor network should direct its resources to the most important feasible missions, reallocating resources appropriately as missions arrive or depart, while taking care not to waste resources on unsuccessful missions. If there are multiple sensors and

multiple missions, it must choose the best matching of sensors to missions. A given sensor may offer different missions varying amounts of information (because of geometry, obstructions, or remaining battery level, for example), or none at all.

A given sensor may offer different missions varying amounts of information (because of geometry, obstructions, or utility requirements), or none at all. Missions, on the other hand, may vary in both importance (*profit*) and difficulty (*demand*), and duration, and these properties need not be correlated. An ongoing surveillance mission may be expensive but of minor importance, whereas an urgent mission for information about one particular spot may be low-demand but very important. In some but not all applications, it may be more beneficial to do one thing well than to do many things badly, that is, to fully satisfy one mission rather than give a small amount of utility to several.

In many applications, partial satisfaction will be no better than zero satisfaction. If the goal of a given mission is to reconstruct the 3D shape of an object, for example, then this may be accomplished with images from two cameras, but an image from just one camera will be useless. Indeed, accepting the single image could actually be harmful since the drain on the sensor's battery could preclude a future mission that might otherwise have been satisfiable. In our model we consider two profit functions: (1) we only receive profit from missions whose demands are fully met, and (2) we consider profits from ones that reached a preset *threshold*. Hence the problem is to choose the "best" assignment of sensors to missions, in the sense that profits from satisfied missions are maximized.

In some networks, there may simply be a static set of long-term missions, in which case the aspect of time may be eliminated. In other settings, mission arrivals and departures may be infrequent, so that for each block of time, sensor assignment can be solved as a static problem. Even in this static setting, our problem is computationally hard to solve optimally.

Thus we use approximation algorithms and heuristics.

A centralized approach to sensor assignment will collect all the relevant information at a central location for decision-making and then distribute assignments. Such an approach can be expensive in terms of communication overhead, however. Another approach is to have nodes[1] make these assignment decisions locally, in a distributed manner, using mission information that is disseminated into the network. While this should decrease communication costs, a centralized algorithm may be able to guarantee better solution.

Since the problem we formulate is (in its most general form) NP-hard even to approximate, we investigate constrained versions for which approximation algorithms exist. First, we bound the number of sensors that may offer contributions to any single mission. This is a reasonable assumption in realistic settings in which sensors have a limited sensing range and the sensors are distributed in such a way as to limit sensing redundancy. Second, we consider a geometric constraint; only sensors within a bounded sensing range from a mission can be assigned to that mission. We also generalize the problem further by allowing a mission to be successful even if its demand is not fully met. We do this by setting a threshold which specifies the minimum fraction of the demand to be met for a mission to succeed. In this case, the mission will not be awarded the full profit but rather a fraction based on its satisfaction level.

Contributions. In this chapter, we consider the problem of assigning directional sensors to missions in wireless sensor networks. We show NP-hardness and give theoretical results for certain problem variants, including a Δ-approximation greedy algorithm and a shifting-based Polynomial-Time Approximation Scheme (PTAS).

The rest of this chapter is organized as follows. Section 11.2 discusses some related

[1] In this chapter we use the terms *sensor* and *node* interchangeably. We use *element* to refer to either a sensor or a mission.

work in sensor networks and in assignment problems. In Section 6.3 we state our network model and formally define the sensor assignment problem and study its computational complexity. In Section 6.4, we study variants of the problem and propose algorithms to solve them. Finally, Section 7.6 concludes the chapter.

6.2 Related Work

Sensor networks. The general problem of choosing sensors to achieve an objective has received sizable attention lately. Several selection objectives have been considered. In [83, 90], for example, the goal is to cover the region using few sensors in order to conserve energy. The techniques used range from dividing the sensors in the network into a number of sets and rotating them, activating one at a time [83], to using Voronoi diagram properties [10] to ensure that the area of a sensor's diagram is as close to its sensing area as possible [90]. Another technique used is delayed sensor activation [76], in which sensors are initially inactive and are activated according to their coverage contribution. In that way the schemes greedily turn on the sensors with highest probability of successfully sensing the areas of interest. Our problem is similar if we consider that covering a certain area is a mission.

Another related problem is to efficiently locate and track targets such as the works in [105] and [68]. In [105], for example, the most informative sensor for tracking a target is chosen based on the concept of information gain. This information is then passed on to the next active sensor, which is chosen by considering the target's expected path. Target localization using acoustic sensors is considered by [68]. An initial active set of two sensors is chosen by exhaustive search, and then additional sensors are added to the active set. The goal there is to minimize the mean squared error of the target location as perceived by the

active sensors. Our work, however, is motivated by contention between multiple missions with varying profit values, and therefore focuses on mission selection rather than sensor selection.

There has also been some work on frameworks for single and multiple mission assignment problems. For example, [28] defines a framework modeling the assignment problem with notions of utility and cost. The goal is to find a solution that maximizes the utility while staying under a predefined budget. In [80], a market-based method is used in which sensors provide information or "goods" which can be purchased while observing certain budgets. The goal is to maximize the amount of goods delivered without exceeding the budget. A survey of sensor selection and assignment problems, including simple theoretical models of the problem, can be found in [86].

The problem we consider here is different from previous work since it considers multiple missions with different priorities. These missions contend for the same set of sensors which calls for resolution mechanisms. We also consider adding energy awareness to our algorithms in order to prolong the network lifetime.

Energy awareness in sensor networks have also been studied in the literature. For example, the authors of [56] propose LEACH which is an energy-efficient communication protocol that uses the concept of clusters. In their scheme they rotate the cluster heads in order to evenly distribute the load among them. This is similar to our energy-aware scheme in which we rotate the sensors assigned to a mission for the same purpose. A survey of energy-efficient scheduling mechanisms in sensor networks can be found in [98].

Algorithms. Although we use the terminology of sensors and missions for concreteness, the problem we are considering can be viewed as a more general problem of resource

allocation. An alternative interpretation regards scheduling jobs on unrelated parallel machines. As in other (maximization) scheduling problems [93], the goal is a schedule that maximizes profit earned from jobs completed, subject to certain constraints. The twist is that each job specifies not the set of *machines* that can perform it, but the set of *families of machines* that can perform it. (A job may be too difficult to be performed by any single machine.) The feasibility constraint is that no machine can be assigned more than one "sub-job".

Our problem also relates to other optimization problems, such as the Bin *Covering* problem, in which the goal is to use a set of items to fill completely as many bins as possible. Sensor-Mission Assignment is a generalization of (weighted) Bin Covering in that an "item" may take up different amounts of space in different "bins". In this way, an analogy can be seen between Sensor-Mission Assignment and the Bin Covering on the one hand and the Multiple Knapsack and the Generalized Assignment (GAP) [39] problems on the other. While the problem we study in this chapter does not involve capacity constraints of GAP, in later work we use a GAP-inspired algorithm to solve a budgeted sensor assignment problem [65].

Combinatorial Auctions. In contrast to conventional auctions, in combinatorial auctions players can bid of sets or *combinations* of items (which may entail exponentially many bids). Given a fixed supply of goods, the goal of the winner determination problem is to maximize revenue earned from the sale of disjoint item combinations. Since this is a difficult problem, much of the research focus has been on AI or algorithm-engineering approaches. (See [35] for a survey.) Another way of understanding our problem is as a combinatorial auction in which bidders are the missions, the items are the sensors *and* the

missions, and each mission places a bid (equal to its own profit) for any set that can be constructed as follows: the set contains the mission itself and some subset of sensors that together satisfy the mission's demand. The language of profits, demands, and edge values, however, allows for a succinct representation of bids.

Many weaker and often fractional models of sensor-mission assignment can be reduced to maximum matching or network flow problems, and thus can be solved optimally in polynomial time [7].

6.3 Overview

In this section we discuss our network model. Then, we formally define the core problem of sensor-mission assignment.

6.3.1 Network Model

In our network model, we assume a set of static sensors pre-deployed in a field. Missions can arrive and depart over time. By a mission, we mean a primitive sensing task that requires information of a certain type, which may be contributed by one or more deployed sensors. Each mission is defined by a specific geographic location. An example of a mission is video monitoring an area of interest. General missions that cover large areas, such as perimeter monitoring, can be divided into multiple missions each having its own location. The deployed sensors are directional in nature and hence each of them can be assigned to a single mission (i.e. directed to one location). The direction of a sensor can be changed when the assignment is changed. A video camera is a good example of such sensors.

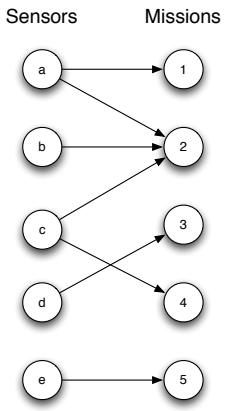


Figure 6.1: Modeling the problem as a bipartite graph

6.3.2 Core Problem Definition

The problem, which we call *Semi-Matching with Demand* (SMD), is modeled as a weighted bipartite graph whose vertex sets consist of sensors $S = \{S_1, ..., S_n\}$ and missions $\{M_1, ..., M_m\}$ (see Figure 6.1). A sensor S_i may be able, depending on its type and location, to provide mission M_j with some data. A positively weighted edge (S_i, M_j) means that S_i is applicable to M_j. The weight of the edge (e_{ij}) indicates the utility (or quality of information) that S_i could contribute to M_j if this pairing were chosen. The utility may vary depending on the sensor's type, location or other properties. Also given is a positive-valued demand d_j associated with each mission M_j, indicating the total utility the mission requires. To simplify the problem we assume that the utility amounts received by a mission are additive. That is the total utility received by a mission is equal to the sum of the utilities provided by sensors assigned to it. While this may be realistic in some settings such as sensing applications in which high-quality measurements can be obtained by e.g. either taking a single high-quality reading or averaging together several lower-quality readings, in others it is not; for our purpose of comparing the different algorithms this assumption is sufficient.

What we seek is a *semi-matching* of sensors to missions, so that (ideally) each mission

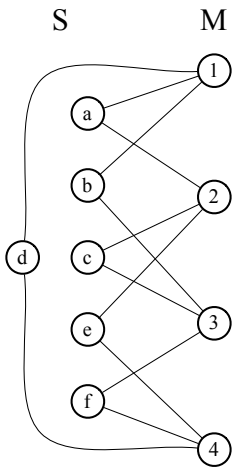


Figure 6.2: Integrality gap instance

demand is satisfied. That is, a sensor may be assigned to at most one of the missions to which it is applicable, but a mission can accept utility from multiple sensors. Of course, satisfying all missions may not be feasible; in general, the goal is to maximize a weighted sum of the *satisfied missions*. We assume there is a profit p_j associated with achieving mission M_j. We then seek to maximize the total satisfied profit.

We start by defining the strict version in which no profit is awarded for a partially satisfied mission. Later (in Section 6.4.3), we relax this requirement and consider a version of the problem in which missions can be partially successful if they reached a minimum threshold of utility.

Unless there is structure to the weights of the sensor-mission edges, for example if they relate to the geometry of node positions, we can assume without loss of generality that each demand is 1. For each mission M_j with demand d_j, simply divide edge value e_{ij} by d_j to obtain an instance with unit demands. Unless otherwise stated, we will assume this normalization henceforth, though it is sometimes convenient to allow for non-unit demands. With this in mind, we define the problem formally.

Instance: A weighted bipartite graph $G = (S, M, P, E)$, where $S = \{S_1, ..., S_n\}$ is a collection of sensors, $M = \{M_1, ..., M_m\}$ is a collection of missions, $P = \{p_1, ..., p_m\}$ is a collection of positive mission profits, and E is a collection of non-negative weights for the edges $S \times M$.

Goal: Find a semi-matching $F \subseteq E$ (no two chosen edges share the same *sensor*), in which $\sum_{M_j \in A} p_j$ is maximized, where $A \subseteq M$ is the set of missions satisfied by F (i.e., $\sum_{(i,j) \in F} e_{ij} \geq 1$ for each $M_j \in A$).

The problem can be formulated as an Integer Program (IP). The IP below employs two sets of decision variables: u_j indicating whether mission M_j is satisfied with the received utility, and x_{ij} indicating whether sensor S_i is assigned to mission M_j. Finding a solution can be seen as a two-step process: decide which missions to satisfy, and then decide how to satisfy them. Each mission M_j has a constraint requiring that the sum of utility received by M_j be at least the value u_j, which is 0 or 1. When $u_j = 0$, this constraint is automatically satisfied. Here is the IP:

Maximize:	$\sum_j p_j u_j$
Such that:	$\sum_{i=1}^{n} x_{ij} e_{ij} \geq u_j$, for each mission M_j,
	$\sum_{j=1}^{m} x_{ij} \leq 1$, for each sensor S_i, and
	$x_{ij} \in \{0, 1\}$, for each variable x_{ij} and $u_j \in \{0, 1\}$, for each variable u_j

Note that if we had not normalized to unit demands, the first constraint would be: $\sum_{i=1}^{n} x_{ij} e_{ij} \geq u_j d_j$.

The corresponding Linear Program (LP), relaxes variable constraints from $\{0, 1\}$ to $[0, 1]$. This allows for fractional profits to be awarded for partially satisfied missions *and* for sensors to be fractionally assigned to multiple missions. This fully fractional version can be solved optimally by Linear Programming. Such an *optimal fractional* solution provides

an upper bound on the true optimal solution value.

Definition 6.3.1. *Let $IP(I)$ indicate the optimal solution value for a given integer program and let $LP(I)$ indicate the optimal solution value of the LP relaxation, both on problem instance I. We will say that the formulation has integrality gap (at least) c for $c \geq 1$, if $c \leq \frac{|LP(I)|}{|IP(I)|}$ for some problem instance I.*

Remark 6.3.1. *The IP above has unbounded integrality gap, since instances can be constructed in which the optimal solution of the corresponding LP $OPT_{LP} = m/2$ and the optimal solution of the IP $OPT_{IP} = 1$, where m is the number of missions. To create such an instance (see Figure 6.2), connect a separate sensor to each pair of missions, so that each mission has $m - 1$ neighbors, and set all demands to $m - 1$ and all profits to 1. Then setting all edge weights to 1/2 will clearly half-satisfy each mission, but only one can be satisfied integrally.*

Definition 6.3.2. *Let $|OPT(I)|$ indicate the optimal solution value for a given problem instance(I), and let $|ALG(I)|$ indicate a corresponding approximate solution value. We will say that an algorithm is a c-*approximation *for $c \geq 1$ if $c \leq \frac{|OPT(I)|}{|ALG(I)|}$ for every problem instance I. We omit I below when it is clear from the context.*

Proposition 6.3.1. *SMD is NP-hard and at least as hard to approximate as* MAXIMUM INDEPENDENT SET (MIS).

Proof. Given an MIS graph $G = (V, E)$, an SMD instance is created with a mission M_v for each $v \in V$, with $d_v = deg(v)$ and $p_v = 1$, and a sensor $S_{u,v}$ for each edge $(u, v) \in E$, which offers utility 1 to missions M_u and M_v. Then the optimal SMD solution yields the optimal maximum independent set (and both share the same solution quality). □

Since MIS cannot be approximated to factor $|V|^{1-\epsilon}$ unless NP=ZPP [53], neither an optimal nor a nontrivial approximation algorithm for SMD is likely to be forthcoming. Therefore we turn in the next section to easier special cases.

6.4 Problem Variants and Algorithms

In this section we discuss a number of problem variants of SMD. First we discuss two special cases which admit approximation algorithms–a degree-bounded version and a geometric variant; then, we study two problem extensions, generalizing to allow a success threshold and generalizing from static to dynamic.

6.4.1 Degree-bounded Approximation Problem

Because of the difficulty of the approximation problem as defined in the previous section, we constrain it in order to render it more tractable. We assume that the problem instance has *bounded degree*, in the following sense. If a sensor S_i makes a non-zero offer to a mission M_j, then say that S_i is M_j's *neighbor*. Then the assumption is that no mission has more than Δ neighbors, for some small constant Δ. We call this problem Δ-SMD.

A simple greedy algorithm (see Algorithm 4) considers missions in decreasing order of profit. For each mission, the algorithm assigns it available sensors in decreasing order of offer utility, until the mission is satisfied. If the mission does not succeed, then all sensors are returned. The running time of this algorithm is $O(m \log m + mn \log n)$ where m is the number of mission and n is the number of sensors. This algorithm provides an approximation guarantee which we prove below.

We note that the algorithm can be given a distributed implementation, proceeding in

Algorithm 1 Δ-Approximation Greedy Algorithm

for each mission M_j in order of decreasing P_j
  for each *still-available* sensor S_i in order of decreasing e_{ij}
    assign S_i to M_j
    if M_j is satisfied **then break**
  if M_j is not satisfied **then**
    return any sensors assigned to it

rounds, as in the distributed greedy algorithm for the Dominating Set problem of [64]. In each round, each pending mission determines whether it is still satisfiable and if so broadcasts its profit value to all other missions within distance $2R_S$. Each pending, satisfiable mission is then assigned its chosen sensors iff it has the highest-profit among such missions in its $2R_S$ neighborhood. Since this is a faithful implementation of the algorithm, we note that the approximation guarantee given below applies to this case.

Definition 6.4.1. *Let a* star *consist of a mission and a* minimally satisfying *set of sensors for it. The sensor set is minimal in the sense that no proper subset would completely satisfy the mission in question. (Notice that a given mission may in general be part of many stars.) Say that a mission is* tight *if it has degree Δ and requires all Δ sensors in order to be satisfied. Two stars* overlap *if they share one or more sensors, if they share a mission, or both, including the case that the stars are identical.*

Proposition 6.4.1. *Algorithm 4 produces a Δ-approximation for the Δ-SMD problem.*

Proof. Let OPT be the set of missions satisfied in some optimal solution (with solution quality $|OPT|$), and let ALG be the missions satisfied by Algorithm 4 (with quality $|ALG|$). We want to show that $|OPT| \leq \Delta \cdot |ALG|$, i.e., that

$$\sum_{M_j \in OPT} p_j \leq \sum_{M_{j'} \in ALG} \Delta \cdot p_{j'} \tag{6.4.1}$$

To prove Ineq. 1, we account for each term p_j on the left-hand side with one of the terms $\Delta \cdot p_{j'}$ on the right-hand side. For each $M_j \in OPT$, say that M_j *charges to* the highest-profit mission $M_{j'} \in ALG$ whose star overlaps with M_j's star, and write $M_j \in ch(M_{j'})$. (There must be one such $M_{j'}$ with $p_j \leq p_{j'}$ since Algorithm 4 satisfies a maximal set of missions, selected in decreasing order of profit.) Then let $M_{j'}$ be an arbitrary mission in ALG. $M_{j'}$ is either tight or not. (Recall Definition 6.4.1.) Suppose tight, in which case that mission has only one star. If $M_{j'} \in OPT$, then only $M_{j'}$ itself charges to $M_{j'} \in ALG$; if $M_{j'} \notin OPT$, then at most Δ stars in OPT can charge to $M_{j'}$ (those that share at least one of its sensors). Now suppose $M_{j'}$ is not tight, so that it contains $\leq \Delta - 1$ sensors. Then at most Δ stars in OPT can charge to $M_{j'}$ (those that share at least one of its sensors, and possibly one that shares its mission). Thus we have

$$\sum_{M_j \in ch(M'_j)} p_j \leq \Delta \cdot p_{j'} \tag{6.4.2}$$

By summing Ineq. 2 over all missions in ALG, we obtain Ineq. 1. □

It is easy to construct an example with $\Delta + 1$ missions to show that the Δ bound is tight: let one mission have profit $1 + \epsilon$ and require all Δ sensors; let the rest of the missions have profit 1 and require one sensor each. We note that such examples can be given even for the geometric settings that we discuss below.

Interestingly, when the number of sensors neighboring a mission is less than or equal to two, then the problem can be solved in polynomial time.

Proposition 6.4.2. 2-SMD *is in P.*

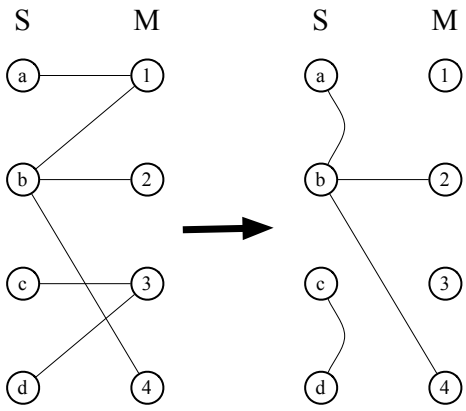


Figure 6.3: Converting SMD to graph.

Proof. We reduce to the (weighted) maximum matching problem (see Figure 6.3). The node set for the resulting graph will consist of the 2-SMD instance's sensors and missions. Whenever a mission M_j will be satisfied only by both its neighbors S_{i1}, S_{i2}, draw an edge (S_{i1}, S_{i2}) with weight equal to the mission profit; whenever a mission M_j will be satisfied by a single sensor S_i, draw an edge (S_i, M_j) with weight equal to the mission profit. Now find a maximum weighted matching in this (non-bipartite) graph in polynomial time. Each selected edge corresponds to a satisfied mission. It is clear that no sensor or mission will be used more than once. The optimal solution values of the matching graph and the SMD are by construction the same. □

Since the graph of a 2-SMD instance is sparse, the maximum weighted matching can be found in time $O(m^2 \log m)$ [43], where m is the number of missions. The time for finding the maximum matching is the dominant component of the running time.

We now relate Δ-SMD and Δ-SET PACKING. These problems turn out to be equivalent for small enough Δ. In the SET PACKING problem, we are given a family of subsets of a universe of elements. Each subset has a positive weight. The goal is to choose a max-weight family of subsets without using any element more than once. Δ-SET PACKING is

the variant of SET PACKING in which each set has at most Δ elements.[2]

Proposition 6.4.3. Δ-SMD *reduces to* Δ-SET PACKING*, when* $\Delta = O(\log nm)$.

Proof. The idea of the reduction is that each star in our SMD instance will become a set in the SET PACKING instance. (Since a given mission may have degree Δ, it can have $O(2^{\Delta})$ many stars. Because of the bound on Δ, however, the resulting SET PACKING instance will be at most polynomially larger than the initial SMD instance size.) Specifically, a star with $s < \Delta$ sensors will become an $s + 1$-element set containing the star's sensors and mission; a star with Δ sensors will become a Δ-element set containing only the star's sensors. Since a mission can have only one tight star of size Δ, the mission need not be included in the resulting star. Choosing a max-weight family of disjoint sets will now be the same as choosing a max-weight set of disjoint stars. □

An existing local search algorithm from Berman [18] gives a ($\frac{\Delta+1}{2}$+ϵ)-approximation for Δ+1-CLAW-FREE MIS, which Δ-SET PACKING reduces to. (ϵ is a running-time parameter, specifically, a $\frac{k}{(k-1)}\frac{\Delta+1}{2}$ approximation can be found in time polynomial in $O(kn)$.) Hence there is a $\frac{\Delta+1}{2}$ approximation for Δ-SMD (for small Δ). It was recently shown [54] that even for the cardinality version, approximating Δ-SET PACKING within a factor better than $\frac{\Delta}{\ln \Delta}$ is NP-hard.

6.4.2 Geometric Problem

We now introduce GEOMSMD, a variant in which geometrically inspired constraints are imposed. First, each sensor and mission now lie at a particular point in the plane. Second, we assume sensors have a bounded sensing range (R_S), i.e., e_{ij} can only be non-zero when

[2]The parameter used is typically k, but we are interested in the case $k = \Delta$.

the distance between S_i and M_j is less than this bound. Without loss of generality, let the sensing range be 1. In this case, every star will lie in a unit disk. We also assume a geometric analog to bounded degree, specifically an upper bound on the number of sensors or missions contained in any unit disk. This constraint will be satisfied automatically if the graph is *drawn in a civilized manner* [61], i.e., so that any two nodes are separated by some minimum distance $\lambda > 0$. In this case, the problem instance will satisfy the Δ-SMD bounded degree requirement for some constant Δ.

The NP-hardness argument below involves the UNIT-DISK MIS (UD-MIS) problem [31], which is an MIS variant in which the problem instance is the *intersection graph* for a set of unit disks lying in the plane. Equivalently, UD-MIS can be defined so that given a set of points in the plane, two points are connected by an edge iff their distance is strictly less than a global constant. We argue that the NP-hardness proof for UD-MIS also applies to a density-bounded UD-MIS.

Proposition 6.4.4. GEOMSMD *is strongly NP-hard.*

Proof. We reduce from PLANAR 3SAT to UD-MIS to GEOMSMD. Given the PLANAR 3SAT instance, we first apply the UD-MIS reduction of [31, 77], which results in a UD-MIS graph and a number k (there is an independent set of size k iff the formula was satisfiable). It can be verified by consulting the proof of [31] that the resulting UD-MIS instance can be drawn with at most O(1) disks per unit square resulting a density-bounded UD-MIS.

Now, we convert the UD-MIS decision-problem instance (G, k) into a GEOMSMD decision-problem instance (G', k), by replacing each disk with a mission at the disk's center, and every maximal intersection of disks with a sensor needed by all of them. Since each mission needs all the sensors lying in its disk in order to be satisfied, k missions can

Algorithm 2 Shifting PTAS (error ϵ)

$k \leftarrow \lceil 2/\epsilon \rceil$; $S = \emptyset$
for each $(i, j) \in [0, k)^2$
  lay the mesh with offset (i, j); $S_{ij} \leftarrow \emptyset$
  for each cell C_t within the mesh
    $S_{ij} \leftarrow S_{ij} \cup opt(C_t)$
  if $val(S) < val(S_{ij})$ **then** $S \leftarrow S_{ij}$

be satisfied iff k independent disks can be chosen. Since in the UD-MIS construction each unit square contains at most $O(1)$ such intersections, in the resulting GEOMSMD instance, each unit square will contain at most $O(1)$ missions and $O(1)$ sensors, and the sensing distance is respected by construction. Thus PLANAR 3SAT is reduced to GEOMSMD. □

Since GEOMSMD is strongly NP-hard, it follows that a *Fully Polynomial-Time Approximation Scheme (FPTAS)* is unlikely for it. A PTAS, however, is possible. To achieve this, we employ the shifting technique originally introduced in [57] which imposes a grid that partitions the region into cells. The portion of the problem instance lying within a given cell has bounded size, which allows the cell to be solved optimally (e.g., by brute force). The portion of the problem instance (in our problem, sensor-mission edges) lying on the boundary of a cell will be discarded, but for a large enough cell size, the perimeter of the cell will be small compared to its area. Although it could happen that much of the problem instance lies near to cell boundaries for a given positioning of the grid, many different grid positions are considered to avoid this problem.

We now give the shifting PTAS[3] (see Algorithm 2), which is similar to the UD-MIS [78] PTAS (following the presentation in [36]). For now, assume for simplicity that all points are inside a square region I whose size is polynomial in the input size. Then for a

[3] Although we focus on the plane, it is easy to extend to a fixed higher dimension D.

desired error bound ϵ, we can choose parameter $c = \lceil 2/\epsilon \rceil$. Now, we lay a square grid on the plane, carving the region into square cells of size $c \times c$. Each integer pair $(i, j) \in [0, c)^2$ corresponds to a possible offset for the coarse-grain grid, i.e., one of the grid positions to be considered. For a given grid position, we eliminate all sensor-mission edges not fully contained within a single cell. Within any cell, there are $O(\Delta c^2)$ sensors and missions; therefore we can find the optimal assignment restricted to that cell by enumerating all $O((\Delta c^2)^{\Delta c^2})$ possible assignments. The solution for a given offset pair (i, j) is the union of the solutions for the individual cells. We compute the solution for each possible offset pair.

If the points lie in an extremely large region, then the method as stated may not run in polynomial time, since there may be exponentially many cells to check. This can be easily fixed. First, notice that there will be at most polynomially many non-empty cells, which can be found by iterating through the point coordinates. For each non-empty cell, we can "grow" it outward, to obtain a maximally non-empty region. Performing this action on every non-empty cell (i.e., Union-Find) produces a polynomial collection of independent regions. Now simply run the original algorithm on each independent region, rather than on the entire space.

Proposition 6.4.5. *Algorithm 2 is a PTAS.*

Proof. Consider the optimal star-set OPT with total profit P_{opt}. By the Shifting Lemma [57], there must be some vertical offset j that crosses a subset of OPT with total profit at most P_{opt}/c. Similarly, there must be some horizontal offset i that crosses a subset of OPT with total profit at most P_{opt}/c. Therefore the union of the cell-optimal solutions for this (i, j) will be within factor $(1 - 1/c)^2 \geq 1 - 2/c \geq 1 - \epsilon$ of the optimal. □

6.4.3 Threshold Problem

We now generalize the core problem so that profit may be awarded fractionally, once a threshold is reached. Each mission M_j is still associated with a positive demand value d_j and a positive profit value p_j. The demand may now be interpreted as the total utility the mission desires. Profit for mission M_j indicates the importance of the mission and is awarded based on the percentage of satisfied demand, but only if this percentage reaches a satisfaction threshold T; p_j is the maximum profit receivable for mission M_j.

The goal is again to maximize total profits. We call this problem Threshold SMD. Threshold-1 SMD (i.e., $T = 1$) is simply the original problem. The problem instance and goal are defined as follows:

Instance: A global threshold $T \in [0, 1]$ and a weighted bipartite graph $G = (S, M, P, D, E)$, where $S = \{S_1, ..., S_n\}$ is a collection of sensors and $M = \{M_1, ..., M_m\}$ is a collection of missions; each mission M_j is associated with a profit $\{p_j\}$ and a demand $\{d_i\}$; each edge in $S \times M$ has an edge-weight e_{ij} indicating utility.

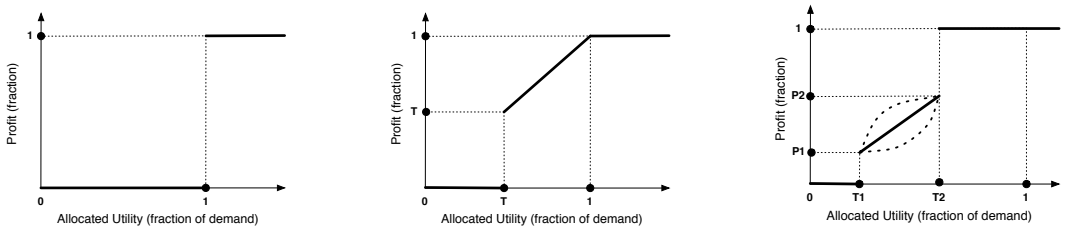
Goal: Find a semi-matching $F \subseteq S \times M$ (no two chosen edges share the same *sensor*), in which $\sum_j p_j(u_j)$ is maximized, where u_j is the total utility received by mission M_j divided by demand d_j. The profit functions are defined as follows:

$$p_j(u_j) = \begin{cases} p_j, & \text{if } u_j \geq 1 \\ p_j \cdot u_j, & \text{if } T \leq u_j < 1 \\ 0, & \text{if } u_j < T \end{cases} \tag{6.4.3}$$

Finally, the IP formulation is the same as the original SMD, except for the objective function:

$$\textit{Maximize:} \quad \sum_j p_j(u_j)$$

Although the objective function is piecewise linear, it is not concave. In fact, it is neither



(a) Strict (b) Fractional with threshold (c) Other functions

Figure 6.4: Modeling missions profits

concave nor convex. Intuitively, the profit function for a single mission is specified by the entire curve in Figure 6.4(b), as the allocated utility to mission M_j (u_j) (represented by the fraction of met demand) ranges from 0 to infinity. In fact, the objective function is only defined from 0 to 1, as a result of the range constraints on the u_j variables in the linear program. Since this is a maximization problem with a non-concave objective, standard LP and IP techniques do not directly apply. It is important to note, that if our profit function were concave on the region $[T, 1]$, it need not be concave overall.

Although we do not consider them in this chapter, there are many other profit functions that may warrant study. For the range lying between profit percentages of 0 and 1, natural options include: linear, full profit only for fully met demands, linear only after crossing a threshold, smooth convex (difficult to optimize) and smooth concave (easy to optimize). A more general family of profit functions depends on two thresholds (see Figure 6.4(c)): before threshold $T1$, there is no partial credit; after threshold $T2$, full credit is received; in between the two thresholds, the profit function could be piecewise convex, linear, or concave, as the application demands, and the profit values at the extremes of this middle range need not equal the corresponding threshold values. Note, however, that strictly speaking, profit functions that go to *full profit* after crossing a threshold can be ignored without loss

of generality, since in this case the threshold is simply a lower demand.

The threshold-based problem generalizes both all-or-nothing profits (with T=1) and fully fractional profits (with T=0). When T=1, profit p_j is received only for fully satisfying mission M_j (see Figure 6.4(a)). In this case, the program reduces to an Integer Program (IP) with mission variable constraints $u_j \in \{0, 1\}$, which is the strict SMD problem. Treating T as part of the problem instance therefore means that this formulation can only be harder than the original strict version (which was shown to be NP-Complete in Section 6.3). Intuitively, lowering the threshold should make the problem easier. The problem, however, is NP-Complete even with threshold 0.

Proposition 6.4.6. *Threshold-0 SMD is* strongly *NP-Complete.*

Proof. We do a reduction from the 3-PARTITION problem [45]. Let $A = \{a_1, ..., a_{3m}\}$ be a multiset of positive integers with $\sum_{a \in A} a = mS$, satisfying $\frac{S}{4} < a < \frac{S}{2}$ for each $a \in A$. The goal in 3-PARTITION is to partition the set A into m triples so that each triple sums to S.

The resulting problem instance has m missions $M_1, ..., M_m$, with demands $d_1 = ... = d_m = S$, unit profits, and $3m$ sensors corresponding to the elements of A. Set all weights for edges (S_i, M_j), to a_i. Then the only way to meet all demands is, by construction, to meet them exactly. Thus the 3-Partition instance has an equal-sum m-partition iff in the resulting SMD instance all missions can be *fully* satisfied. Finally, it is clear that an assignment that satisfies all missions can be checked in polynomial time. □

We now discuss how the greedy algorithm given above in Algorithm 4 adapts to the threshold setting (see Algorithm 3). The algorithm will repeatedly satisfy the most *currently profitable* mission, i.e. the mission that can be satisfied with the greatest profit, using the currently available sensors. If $S' \subset S$ is the set of not-yet-assigned sensors (initially

Algorithm 3 Greedy Algorithm for Threshold-SMD

while true
  for each available mission M_j
    $u_j \leftarrow \sum_{S_i \text{unused}} e_{ij}$
  $j \leftarrow \arg\max_j p_j(u_j)$
  if $p_j(u_j) = 0$ **then break**
  $u_j \leftarrow 0$
  for each unused S_i in decreasing order of e_{ij}
    if $u_j \geq d_j$ or $e_{ij} = 0$ **then break**
    assign S_i to M_j
    $u_j = u_j + e_{ij}$

$S' = S$) and $u_j = \sum_{S_i \in S'} e_{ij}$, then the profit currently achievable by mission M_j is $p_j(u_j)$. (Of course, it may be that not all sensors are needed to achieve this profit; conversely, if the demand threshold is not met, this profit is 0.) The algorithm repeatedly select a mission M_j of maximum current profitability, and then satisfies it with available sensors (which are removed from S'), in order of decreasing contribution value e_{ij}, until either M_j is *fully* satisfied or all sensors with non-zero offers to M_j have been used. When there are no remaining missions with non-zero current profitability, the algorithm completes. The running time of the algorithm as written is $O(n(m + \log n))$ but it is easy to improve this to $O(mn \log n)$ by updating the u_j values over time rather than computing them from scratch. This greedy algorithm has the same Δ-approximation performance guarantee.

Proposition 6.4.7. *Algorithm 3 produces a Δ-approximation for the problem.*

Proof. The proof of Theorem 6.4.1 goes through essentially unchanged. □

When $T = 1$, this algorithm produces the same result as the simpler greedy algorithm of Algorithm 4. We note that this algorithm also can be given a distributed rounds-based implementation, similar to that for Algorithm 4. The only difference is that the value each

mission broadcasts to its distance $2R_S$ neighbors is now its current profitability value, rather than its profit value.

For geographic settings with limited sensing range, we can also extend the approximation scheme (PTAS) of the previous subsection to the threshold problem (see [88]). Although it is theoretically efficient, we found in our experiments that achieving performance competitive with the greedy algorithm's requires an unreasonable time, and so we do not include its results in this chapter.

6.4.4 Dynamic Problem Model

We now provide an orthogonal generalization of the original problem in terms of time to model more realistic scenarios in which missions arrive and depart over time. The problem statement is the same, except that now each mission is associated with a start time and an end time. A mission's demand and maximum profit are constant over time. Awarded profit for a mission is computed at each discrete timestep, based on the satisfaction level at that instant. Total profit for a mission is simply the sum of the instantaneous profits. We do not require that a mission's demand be met over its entire lifetime in order to receive profit. Our profit model is in this sense fractional *in terms of time*. The dynamic version is thus given by essentially the same mathematical program given above except that each variable now has an additional time index.

As a generalization of the static problem, previous hardness results apply also to the dynamic version. Indeed, a natural strategy for the dynamic problem is to solve the static problem at each timestep.

6.5 Conclusion

In this chapter, we introduced a sensor-mission matching problem. We analyzed its complexity, defined constrained versions, and presented approximation algorithms for them. We showed that it is strongly NP-hard, even in the special case of zero success threshold. As such, we turned to approximation algorithms and heuristics. We considered a greedy centralized algorithm and several distributed schemes, which we then adapted to the dynamic setting.

In our algorithms and experiments, we make the simplifying assumption that utilities from different sensors can be combined additively. Clearly this assumption applies to some but far from all settings. Building on the work presented here, we are currently investigating joint utility functions in which the combined utility of multiple sensors could be either greater than or less than the sum of the individual utilities. The joint utility setting is more complex and will require substantially revised algorithms. The algorithms presented here represent a first step towards a solution to this more general problem, but we also believe the additive utility setting is interesting in its own right.

Chapter 7

Frugal Sensor Assignment

7.1 Introduction

In this chapter, we present simplified problem formulations of sensor-task assignment problems, before extending both to more realistic settings and notions of frugality in later chapters. We again examine situations in which sensors offer different utility amounts to different missions, which are each associated with profit and demand levels. In this chapter, we examine new sensor-assignment problems motivated by frugality, i.e., the conservation of resources. We consider two broad classes of environments: static and dynamic. The *static* setting is motivated by situations in which different users are granted control over the sensor network at different times. During each time period, the current user may have many simultaneous missions. While the current user will want to satisfy as many of these as possible, sensing resources may be limited and expensive, both in terms of equipment and operational cost. In some environments, replacing batteries may be difficult, expensive, or dangerous. Furthermore, a sensor operating in active mode (i.e., assigned to a mission)

may be more visible than a dormant sensor, and so in greater danger of tampering or damage. Therefore, we give each mission in the static problem a budget so that no single user may overtax the network and deprive future users of resources. This budget serves as a constraint in terms of the amount of resources that can be allocated to a mission regardless of profit.

Our second environment is a *dynamic* setting in the sense that missions arrive over time and have different durations. In these cases explicit budgets may be too restrictive because we must react to new missions given our current operating environment, that is, the condition of the sensors will change over time. Instead, we use battery lifetime as a means of discouraging excessive assignment of sensors to any one mission, evaluating trade-offs between the relative value of a given assignment and the expected profit earned with each sensor (given the mission statistics).

In the dynamic setting we consider two cases. First, we assume no advanced knowledge of the target network lifetime, i.e. we do not know for how long the network will be required to operate. We call this the *general dynamic setting*. Second, we consider the case in which we have knowledge of the target network lifetime, i.e. the network is needed for a finite duration. We call this the *dynamic setting with a time horizon*. In our algorithms, we adjust the aggressiveness with which sensors accept new missions, based on trade-offs between target network lifetime, remaining sensor energy, and mission profit, rather than using hard budgets.

Our contributions. We consider several sensor-assignment problems, in both budget-constrained static settings and energy-constrained dynamic settings. In the static setting, we give an efficient greedy algorithm and a multi-round proposal algorithm whose subroutine solves a Generalized Assignment Problem (GAP). We also give in the appendix an optimal

algorithm for a simplified one-dimensional setting. In the dynamic setting, we develop distributed schemes that adjust sensors' eagerness to participate in new missions based on their current operational status and the target network lifetime (if known).

The rest of this chapter is organized as follows. Section 11.2 presents some related work. In Section 7.3.1 we define our network model, and in Section 7.3 we review the sensor-mission assignment problem and summarize its variations. In Sections 7.4 and 7.5 we propose schemes for the static and dynamic settings. Finally, Section 7.6 concludes the chapter.

7.2 Related Work

Sensor networks. Assignment and selection problems have received sizable attention, in both experimental and theoretical communities. A survey of the different sensor selection and assignment schemes can be found in [86].

The authors of [76, 83, 90], for example, solve the coverage problem, which is related to the assignment problem. They try to use the smallest number of sensors in order to conserve energy. The techniques used range from dividing nodes[1] in the network into a number of sets and rotating through them, to activating one set at a time [83], to using Voronoi diagram properties to ensure that the area of a node's region is as close to its sensing area as possible [90]. Sensor selection schemes have also been proposed to efficiently locate and track targets. For example, [105] uses the concept of information gain to select the most informative sensor to track a target. [68] attempts target localization using acoustic sensors. The goal is to minimize the mean squared error of the target location as perceived by the active nodes. In these works, however, sensors are typically being chosen to work

[1] We use the terms *node* and *sensor* interchangeably.

together to perform a single task, such as covering an area or tracking a target, whereas our work here is essentially concerned with competition among many simultaneous missions.

In wireless sensor networks, there has been some work in defining frameworks for single and multiple mission assignment problems. For example, [28] defines a framework for modeling the assignment problem by using the notions of utility and cost. The goal is to find a solution that maximizes the utility while staying under a predefined budget. Here again, though, the sensors are working cooperatively on a single task, such as monitoring toxicity levels in an area. In [80], a market-based modeling approach is used, with sensors providing information or "goods". In this case, however, the motivation is more about testing auction mechanisms than in optimization algorithms.

Algorithms. The Generalized Assignment Problem (GAP) [33] is a generalization of the Multiple Knapsack Problem, in which the weight and value of an item may vary depending on the bin in which it is placed. There is a classical FPTAS [95] for the core Knapsack problem which performs a dynamic programming (DP) procedure on a discretization of the problem input. If the knapsack budget value e.g. is not too large, then the DP can find the optimal solution in polynomial time. A stricter version of the static sensor-assignment problem was formalized as Semi-Matching with Demands (SMD) in [12, 87]. In that formulation, profits are awarded only if a certain utility threshold is met but no budgets are considered. The static problem we study here is a common generalization of these two previous problems, incorporating both budgets and a profit threshold.

Although we use the terminology of sensors and missions for concreteness, the problem we are considering can be viewed as a more general problem of resource allocation. An alternative interpretation regards scheduling jobs on unrelated parallel machines. As

in other (maximization) scheduling problems [93], the goal is a schedule that maximizes profit earned from jobs completed, subject to certain constraints. The twist is that each job specifies not the set of *machines* that can perform it, but the set of *families of machines* that can perform it. (A job may be too difficult to be performed by any single machine.) The feasibility constraint is that no machine can be assigned more than one "sub-job".

Our problem also relates to other optimization problems, such as the Bin *Covering* problem, in which the goal is to use a set of items to fill completely as many bins as possible. Sensor-Mission Assignment is a generalization of (weighted) Bin Covering in that an "item" may take up different amounts of space in different "bins". In this way, an analogy can be seen between Sensor-Mission Assignment and the Bin Covering and the Multiple Knapsack and the Generalized Assignment [39] problems.

Many weaker and often fractional models of sensor-mission assignment can be reduced to maximum matching or network flow problems, and thus can be solved optimally in polynomial time [7].

7.3 Sensor-Mission Assignment Problems

With multiple sensors and multiple missions, sensors should be assigned in the "best" way. This goal is shared by all the problem settings we consider. There are a number of attributes, however, that characterize the nature and difficulty of the problem. In this section, we start by defining our network model. Then, we briefly enumerate the choices available in defining a particular sensor-mission assignment problem. In all settings we assume that a sensor can be assigned to one mission only, motivated e.g. by directional sensors such as cameras.

7.3.1 Network Model

In our network model, we assume a set of static sensors pre-deployed in a field in a uniformly random manner. Missions can arrive and depart over time. By a mission, we mean a primitive sensing task that requires information of a certain type, which may be contributed by one or more deployed sensors. Each mission is defined by a specific geographic location. (While mission locations could be chosen from any distribution in principle, in the simulation described below mission locations are chosen uniformly at random.) An example of a mission is video monitoring an area of interest. General missions that cover large areas, such as perimeter monitoring, can be decomposed into multiple missions, each with its own (discrete) location. The deployed sensors are directional in nature, and hence can be assigned to missions individually. Sensors have limited sensing range R_s; a sensor provides no utility to any mission that is located beyond this distance. The sensors are equipped with wireless communication equipment and have limited communication range R_c. The communication range is defined as the maximum distance over which two sensors can directly communicate.

7.3.2 Static Setting

First consider the static setting. Given is a set of sensors $S_1, ..., S_n$ and a set of missions $M_1, ..., M_m$. Each mission is associated with a utility demand d_j, indicating the amount of sensing resources needed, and a profit p_j, indicating the importance of the mission. Each sensor-mission pair is associated with a utility value e_{ij} that mission M_j will receive if sensor S_i is assigned to it. This can be a measure of the quality of information that a sensor can provide to a particular mission. To simplify the problem, we assume that the utility values e_{ij} received by a mission M_j are additive. While this may be realistic in some

settings, in others it is not. For example, in a localization application, the utility provided by sensors is not additive as it depends on the relative angles by which they view the target. For our purpose of comparing the different algorithms, however, this utility model is sufficient. Finally, a budgetary restriction is given in some form, either constraining the entire problem solution or constraining individual missions as follows: each mission has a budget b_j, and each potential sensor assignment has cost c_{ij}. All the aforementioned values are positive reals, except for costs and utility, which could be zero. The most general problem is defined by the following mathematical program (MP) **P**:

$$\begin{array}{ll} \textit{maximize:} & \sum_{j=1}^{m} p_j(y_j) \\ \textit{such that:} & \sum_{i=1}^{n} x_{ij} e_{ij} \geq d_j y_j, \text{ for each } M_j, \\ & \sum_{i=1}^{n} x_{ij} c_{ij} \leq b_j, \text{ for each } M_j, \\ & \sum_{j=1}^{m} x_{ij} \leq 1, \text{ for each } S_i, \\ & x_{ij} \in \{0,1\}\ \forall i,j \text{ and} \\ & y_j \in [0,1]\ \forall j \end{array}$$

A sensor can be assigned ($x_{ij} = 1$) at most once. Profits are received per mission, based on its satisfaction level y_j. Note that y_j corresponds to u_j/d_j within the range [0,1] where $u_j = \sum_{i=1}^{n} x_{ij} e_{ij}$. With *strict* profits, a mission receives exactly profit p_j iff $u_j \geq d_j$. With *fractional* profits, a mission receives a fraction of p_j proportional to its satisfaction level y_j and at most p_j. More generally, profits can be awarded fractionally, after reaching a fractional satisfaction threshold T:

$$p_j(u_j) = \begin{cases} p_j, & \text{if } u_j \geq d_j \\ p_j \cdot u_j/d_j, & \text{if } T \leq u_j/d_j \\ 0, & \text{otherwise} \end{cases}$$

When $T = 1$, program **P** is an integer program; when $T = 0$, it is a mixed integer program with the decision variables x_{ij} still integral.

The edge values e_{ij} may be arbitrary non-negative values, or may have additional structure. If sensors and missions lie in a metric space, such as the line or plane, then edge values may be based in some way on the distance D_{ij} between sensor S_i and mission M_j. In the *binary sensing* model, e_{ij} is equal to 1 if distance D_{ij} is at most the sensing range R_S, and 0 otherwise. In another geometric setting, e_{ij} may vary smoothly based on distance, according to a function such as $1/(1 + D_{ij})$.

Similarly, the cost values c_{ij} could be arbitrary or could exhibit some structure: the cost could depend on the sensor involved, or could e.g. correlate directly with distance D_{ij} to represent the difficulty of moving a sensor to a certain position. It could also be unit, in which case the budget would simply constrain the number of sensors.

Even if profits are unit, demands are integers, edge values are 0/1 (though not necessarily depending on distance), and budgets are infinite, then this problem is NP-hard and as hard to approximate as Maximum Independent Set [12]. If we also add the restriction that sensors and missions lie in the plane and that 0/1 edge utility depends on distance (i.e., the edges form a bipartite unit-disk graph), then solving the problem optimally remains NP-hard [12].

We show in the Appendix, however, that in certain one-dimensional settings, the problem can be solved optimally in polynomial-time by dynamic programming. This problem setting is motivated by environments such as coastlines and national border surveillance.

7.3.3 Dynamic Setting

In the dynamic setting, we do away with explicit budgets for missions and explicit costs for sensor assignments. What constraints the assignment problem now is the limited energy that sensors have. The other essential change is the introduction of a time dimension, on

a discrete time model in which time is divided into discrete timeslots. In this setting, each sensor has a battery size B, which means that it may only be used for at most B timeslots over the entire time horizon. Missions may arrive at (the start of) any timeslot and may last for any discrete duration.

If a sensor network is deployed with no predetermined target lifetime, then the goal may be to maximize the profit achieved by each sensor during its own lifetime. However, if there is a finite target lifetime for the network, then the goal is to earn the maximum total profits over the entire time horizon. We assume that the profit for a mission that lasts for multiple timeslots is the sum of the profits earned over all timeslots during the mission's lifetime.

The danger in any particular sensor assignment is then that the sensor in question might somehow be better used at a later time. Therefore the challenge is to find a solution that competes with an algorithm that knows the characteristics of all future missions before they arrive. The general dynamic problem is specified by the following mathematical program (MP) **P'**:

$$
\begin{array}{ll}
\textit{maximize:} & \sum_t \sum_{j=1}^{m} p_j(y_{jt}) \\
\textit{such that:} & \sum_{i=1}^{n} x_{ijt} e_{ij} \geq d_j y_{jt}, \text{ for each } M_j \text{ and } t, \\
& \sum_{j=1}^{m} x_{ijt} \leq 1, \text{ for each } S_i \text{ and time } t, \\
& \sum_t \sum_{j=1}^{m} x_{ijt} \leq B, \text{ for each } S_i, \\
& x_{ijt} \in \{0,1\}\ \forall i,j,t \text{ and} \\
& y_{jt} \in [0,1]\ \forall j,t
\end{array}
$$

If *preemption* is allowed, i.e., a new mission is allowed to preempt an ongoing mission and grab some of its sensors, then in each timeslot we are free to reassign currently used sensors to other missions based on the arrival of new missions, without reassignment costs. In this case, a long mission can be thought of as a series of unit-time missions, and so the sensors and missions at each timeslot form an instance of the NP-hard static problem. If

preemption is forbidden, then the situation for the online algorithm is in a way simplified. If we assume without loss of generality that no two missions will arrive at exactly the same time, then the online algorithm can focus on one mission at a time. Nonetheless, the dynamic problem remains as hard as the static problem, since a reduction can be given in which the static missions are identified with dynamic missions of unit length, each starting ϵ after the previous one. In fact, we can give a stronger result, covering settings both with and without preemption.

Proposition 7.3.1. *There is no constant-competitive algorithm for the online dynamic problem,* even assuming that all missions are unit-length and non-overlapping.

Proof. First consider the problem with $B = 1$, i.e. sensors with a battery lifetime of one timeslot. Suppose at time 1, there is mission M_1 with profit $p_1 = \epsilon$ requiring sensor S_1 (i.e., otherwise unsatisfiable); we must choose the assignment because there may be no further missions, yet at time 2 there may be a M_2 with $p_2 = 1$. This yields the negative result for $B = 1$. Next suppose $B = 2$. Then consider the following example: at time 1, M_1 with $p_1 = \epsilon$ requires S_1, so we must assign because there may be no future missions; at time 2, M_2 with $p_2 = \exp(\epsilon)$ requires S_1, so we must assign for the same reason; at time 3, M_3 with $p_2 = \exp(\exp(\epsilon))$ requires S_1, but it is now empty, and the algorithm fails. This construction can be extended to arbitrary B. □

As a generalization of the static problem, the offline version of the dynamic problem is again NP-hard and as hard as Independent Set to approximate; moreover, the general online version cannot be solved with a competitiveness guarantee.

7.4 Algorithms for the Static Setting

In this section we describe two algorithms to solve the static-assignment problem: *Greedy* and *Multi-round Generalized Assignment Problem (MRGAP)*. The former requires global knowledge of all missions to run and hence is considered centralized, whereas the latter can be implemented in both centralized and distributed environments, a benefit in the sensor network domain. We note here (see the Appendix) that in the one-dimensional setting referred to above, the problem can be solved optimally in $O(mn^2)$ time. This can be of practical interest in monitoring applications such as guarding a national border or coastline.

7.4.1 Greedy

The first algorithm we consider (Algorithm 4) is a greedy algorithm that repeatedly attempts the highest-potential-profit untried mission. Because fractional profits are awarded only beyond the threshold percentage T, this need not be the mission with maximum p_j. For each such mission, sensors are assigned to it, as long the mission budget is not yet violated, in decreasing order of cost-effectiveness, i.e., the ratio of edge utility for that mission and the sensor cost. The running time of the algorithm is $O(mn(m + \log n))$. No approximation factor is given for this efficiency-motivated algorithm since, even for the first mission selected, there is no guarantee that its feasible solution will be found. This by itself is, after all, an NP-hard 0/1 Knapsack problem.

7.4.2 Multi-round GAP (MRGAP)

The idea of the second algorithm (shown in Algorithm 5) is to treat the missions as knapsacks that together form an instance of the Generalized Assignment Problem (GAP). The

Algorithm 4 Greedy

```
1:  while true do
2:      for each available mission M_j do
3:          u_j ← Σ_{S_i unused} e_ij
4:      end for
5:      j ← arg max_j p_j(u_j)
6:      if p_j(u_j) = 0 then
7:          break
8:      end if
9:      u_j ← 0
10:     c_j ← 0
11:     for each unused S_i in decreasing order of e_ij/c_ij do
12:         if u_j ≥ d_j or e_ij = 0 then
13:             break
14:         end if
15:         if c_j + c_ij ≤ b_k then
16:             assign S_i to M_j
17:             u_j ← u_j + e_ij
18:             c_j ← c_j + c_ij
19:         end if
20:     end for
21: end while
```

strategy of this algorithm is to find a good solution for the problem instance *when treated as GAP*, and then to do postprocessing to enforce the lower-bound constraint of the profit threshold, by removing missions whose satisfaction percentage is too low. Releasing these sensors may make it possible to satisfy other missions, which suggests a series of rounds. In effect, missions not making good progress towards satisfying their demands are precluded from competing for sensors in later rounds.

Cohen et al. [33] give an approximation algorithm for GAP which takes a knapsack algorithm as a parameter. If the knapsack subroutine has approximation guarantee $\alpha \geq 1$, then the Cohen GAP algorithm offers an approximation guarantee of $1 + \alpha$. We use the standard knapsack FPTAS [95], which yields a GAP approximation guarantee of $2 +$

Algorithm 5 Multi-Round GAP (MRGAP)

```
1: while true do
2:    initialize set of remaining missions M ← {M_1 ... M_m}
3:    initialize global threshold T ← 0.05
4:    for t = 0 to T, step 0.05 do
5:       run the GAP algorithm of [33] on M and the unassigned sensors
6:       in the resulting solution, release any superfluous sensors
7:       update the missions' demands and budgets accordingly
8:       if M_j's satisfaction level is < T, for any j then
9:          release all sensors assigned to M_j
10:         M ← M − {M_j}
11:      end if
12:      if M_j is completely satisfied OR has no remaining budget, for any j then
13:         M ← M − {M_j}
14:      end if
15:      if M = ∅ or all sensors have been assigned then
16:         break
17:      end if
18:   end for
19: end while
```

ϵ. Because GAP does not consider lower bounds on profits for the individual knapsacks, which is an essential feature of our sensor-assignment problem, we enforce it using the postprocessing step.

The algorithm works as follows. The threshold is initialized to a small value, e.g. 5%. In each round, a GAP solution is found based on the remaining sensors and missions. After each round, missions not meeting the threshold are removed, and their sensors are released. Any sensors assigned to a mission that has greater than 100% satisfaction, and which can be released without reducing the percentage below 100%, are released. (Such sensors are *superfluous*.) Sensors assigned to missions meeting the threshold remain assigned to those missions. These sensors will not be considered in the next round, in which the new demands and budgets of each mission will become the remaining demand and the remaining budget

of each one of them. Finally, the threshold is incremented, with rounds continuing until all sensors are used or all missions have succeeded or been removed. The GAP instance solved at each round is defined by the following linear program:

$$\begin{array}{ll} \textit{maximize:} & \sum_j \sum_i p_{ij} x_{ij} \text{ (with } p_{ij} = p_j \cdot e_{ij}/\hat{d}_j) \\ \textit{such that:} & \sum_{S_i \text{unused}} x_{ij} c_{ij} \leq \hat{b_j}, \text{ for each remaining } M_j, \\ & \sum_{M_j \text{remaining}} x_{ij} \leq 1, \text{ for each unused } S_i, \text{ and} \\ & x_{ij} \in \{0,1\}\ \forall i,j \end{array}$$

Here $\hat{d}_j$ is the remaining demand of M_j, that is, the demand minus utility received from sensors assigned to it during previous rounds. Similarly, $\hat{b_j}$ is the remaining budget of M_j. The concepts of demand and profit are encoded in the gap model as $p_{ij} = p_j \cdot e_{ij}/\hat{d}_j$. This parameter represents the fraction of demand satisfied by the sensor, scaled by the priority of the mission. In each GAP computation, we seek an assignment of sensors that maximizes the total benefit brought to the demands of the remaining mission.

One advantage of MRGAP is that it can be implemented in a distributed fashion. For each mission there can be a sensor, close to the location of the mission, that is responsible for running the assignment algorithm. Missions that do not contend for the same sensors can run the knapsack algorithm simultaneously. If two or more missions contend for the same sensors, i.e. they are within distance $2r$ of one other, then synchronization of rounds is required to prevent them from running the knapsack algorithm at the same time. To do this, one of the missions (e.g. the one with the lowest id) can be responsible for broadcasting a synchronization message at the beginning of each new round. However, since r is typically small compared to the size of the field, we can expect many missions to be able to do their computations simultaneously.

The total running time of the algorithm depends on the threshold T and the step value chosen, as well as on the density of the problem instance, which will determine to what

degree the knapsack computations in each round can be parallelized.

7.5 Algorithms for the Dynamic Setting

We have shown in Section 7.3.3 that the dynamic problem is NP-hard to solve and that without assuming any additional constraints there are no competitive solutions that can provide guaranteed performance. In this section, we therefore propose heuristic-based schemes to solve the dynamic sensor-mission assignment problem. These schemes are similar in operation to the dynamic proposal scheme proposed in [87], but with a new focus. Rather than maximizing profit by trying to satisfy all available missions, we focus here on maximizing the profit over network lifetime by allowing the sensors to refuse participation in missions they deem not worthwhile.

We deal with missions as they arrive. A *mission leader*, a node that is close to the mission's location, is selected for each mission. Finding the leader can be done using a geographic-based routing techniques [69]. The mission leaders are informed about their missions' demands and profits by a base station. They then run a local protocol to match nearby sensors to their respective missions. Since the utility a sensor can provide to a mission is limited by sensing range, only sensors that are close to the mission need to be considered. The leader advertises its mission information (demand and profit) to the nearby nodes. The number of communication hops the advertisement message is sent over to reach all applicable sensors depends on the relationship between communication range, R_c, and sensing range, R_s.

When a nearby sensor hears such an advertisement message, it makes a decision either to propose to the mission and become eligible for selection by the leader or to ignore the advertisement. The decision is based on the current state of the sensor (and the network if

known) and on potential contribution to mission profit that the sensor would be providing. We assume knowledge of the (independent) distributions of the various mission properties (namely, demand, profit and lifetime), which can be learned from historical data. To determine whether a mission is worthwhile, a sensor considers a number of factors:

- the mission's profit, relative to the maximum profit
- the sensor's utility to the mission, relative to the mission's demand
- the sensor's remaining battery level
- the remaining target network lifetime, if known

After gathering proposals from nearby sensors, the leader selects sensors based on their utility offers until it is fully satisfied or there are no more sensor offers. The mission (partially) succeeds if it reaches the success threshold; if not, it releases all sensors.

Since we assume all distributions are known, the share of mission profit potentially contributed by the sensor (i.e. if its proposal is accepted) can be compared to the expectation of this value. Based on previous samples, we can estimate the expected mission profit $E[p]$ and demand $E[d]$. Also, knowing the relationship between sensor-mission distance and edge utility, and assuming a uniform distribution on the locations of sensors and missions, we can compute the expected utility contribution $E[u]$ that a sensor can make to a typical mission *in its sensing range*. We use the following expression to characterize the expected partial profit a sensor provides to a typical mission:

$$E\left[\frac{u}{d}\right] \times \frac{E[p]}{P} \tag{7.5.1}$$

We consider two scenarios. In the first, the target network lifetime is unknown: we do not know for how long will the network be needed. In this case, sensors choose missions

that provide higher profit than the expected value and hence try to last longer in anticipation of future high profit missions. In the second, the target network lifetime is known: we know the duration for which the network will be required. In this case, sensors take the remaining target network lifetime into account along with their expected lifetime when deciding whether to propose to a mission. In the following two subsections we describe solutions to these two settings.

7.5.1 Energy-aware Scheme

In this scheme, the target lifetime of the sensor network is unknown. For a particular sensor and mission, the situation is characterized by the actual values of mission profit (p) and demand (d) and by the utility offer (u), as well as the fraction of the sensor's remaining energy (f). For the current mission, a sensor computes this value:

$$\frac{u}{d} \times \frac{p}{P} \times f \tag{7.5.2}$$

Each time a sensor becomes aware of a mission, it evaluates expression (2). It makes an offer to the mission only if the value computed is greater than expression (1). By weighting the actual profit of a sensor in (2) by the fraction of its remaining battery value, the sensors start out eager to propose to missions but become increasingly selective and cautious over time, as their battery levels decrease. The lower a sensor's battery gets, the higher relative profit it will require before proposing to a mission. Since different sensors' batteries will fall at different rates, in a dense network we expect that most feasible missions will still receive enough proposals to succeed.

7.5.2 Energy- and Lifetime-aware Scheme

If the target lifetime of the network is known, then it can be pre-configured in all sensors before deployment. In this case, sensors can take the target lifetime into account when making their proposal decisions. To do this, a sensor needs to compute the *expected occupancy time* t_α, the amount of time a sensor expects to be assigned to a mission during the remaining target network lifetime. To find this value we need to determine how many missions a given sensor is expected to see. Using the distribution of mission locations, we can compute the probability that a random mission lies within a given sensor's range. Combining this with the remaining target network lifetime and arrival rate of missions, we can find the expected number of missions to which a given sensor will have the opportunity to propose. Thus if the arrival rate and the (*independent*) distributions of the various mission properties are known, we can compute t_α as follows:

$$t_\alpha = \tau \times \lambda \times g \times \gamma \times E[l]$$

where:

- τ is the remaining target network lifetime, i.e. the initial target network lifetime minus current elapsed time
- λ is the mission arrival rate
- $g = \pi {R_s}^2 / A$ is the probability that a given mission location (chosen uniformly at random) lies within sensing range, R_s is the sensing range, and A is the area of the deployment field
- $E[l]$ is the expected mission lifetime

- γ is the probability that a sensor's offer is accepted[2].

For each possible mission, the sensor now evaluates an expression which is modified from (2). The sensor considers the ratio between its remaining lifetime and its expected occupancy time. If t_b is the amount of time a sensor can be actively sensing, given its current energy level, the expression then becomes:

$$\frac{u}{d} \times \frac{p}{P} \times \frac{t_b}{t_\alpha} \tag{7.5.3}$$

If the value of expression (4) is greater than that of expression (1), then the sensor proposes to the mission. Moreover, if the sensor's remaining target lifetime is greater than its expected occupancy time, the sensor proposes to *any* mission since in this case it expects to survive until the end of the target time. The effect on the sensor's decision of weighting the mission profit by the ratio (t_b/t_α) is similar to the effect weighting the fraction of remaining energy (f) had in expression (2); all things being equal, less remaining energy makes a sensor more reluctant to propose to a mission. As the network approaches the end of its target lifetime, however, this ratio will actually increase, making a sensor more willing to choose missions with profits less than what it "deserves" in expectation. After all, there is no profit at all for energy conserved past the target lifetime.

7.6 Conclusion

In this chapter, we defined new sensor-assignment problems motivated by frugality and conservation of resources, in both static and dynamic settings. We proposed schemes to match sensing resources to missions in both settings and evaluated these schemes through

[2] *Computing* this value would imply a circular dependency; it was chosen a priori to be 0.25).

simulations.

Although we do not consider them in this chapter, there are many other profit functions that may warrant study. Obviously zero utility should yield zero profit, and full utility full profit. For the shape of the profit curve lying between these two extremes, natural options include: linear, full profit only for fully met demands, linear only after crossing a threshold, smooth convex (often difficult to maximize) and smooth concave (often easier to maximize).

More sophisticated utility models, taking into consideration not just physical attributes such as distance but also some sort of semantics, could be developed. Recent work by Bisidikian [23], for example, considered the effects of sensor sampling models and "quality of information". Above, we based the potential utility of a sensor-missing pairing on their separating distance. However, in the case of video sensors, for example, the closest sensor to an event might not be the best candidate for selection because its view of the field is obstructed or it cannot provide sufficient frame rate. Also, regarding the value of utility from multiple sensors, we make the simplifying assumption of additive utilities, which is realistic in some but clearly not all scenarios. In 3-d reconstruction, for example, the benefit of having two viewing angles separated by 45 degrees may be much greater than two views from similar locations. In ongoing work, we are investigating models that allow for such non-additive joint utility.

Finally, an implicit assumption we make in this chapter is that all sensors can provide the same type of information, possibly with varying quality. In more realistic scenarios, however, we can expect heterogeneous networks with different types of sensors. In this case missions may have complex demands, involving multiple data types, rather than demands represented by a single scalar value. Part of our future research plan includes extending

this work to support such multimodal settings.

Appendix: 1-d Static Setting

In 1-d settings such as border surveillance, sensors and missions lie on the line, and edge weights e_{ij} depend in some way on the distance between S_i and M_j. If $e_{ij} = 1/(1 + D_{ij})$, for example, then the 1-d problem is strongly NP-hard, even with no budget constraint, since the known 3-Partition reduction [88] also works in 1-d. To adapt the reduction to 1-d, place all missions at one point on the line, and choose the sensor locations so that the resulting edge weights equal the corresponding 3-Partition element values.

If we adopt a binary sensor model, however, then the budget-constrained 1-d SMD problem is in *P*. We give a polynomial-time algorithm for a 1-dimensional, budget-constrained version of SMD in which edge-weights are 0/1 based on sensing range and the budget limits the *total number of sensors* that may be used. In this setting, $e_{ij} = 1$ if $D_{ij} \leq r$ and 0 otherwise. We may therefore assume without loss of generality that the demands d_j are integral, but profits p_j may be arbitrary positive values. Assignments and profits are both non-fractional. For now, assume budgets are infinite.

We first observe that if r is large enough so that $e_{ij} = 1$ for all i, j then this is an instance of the 0/1 Knapsack problem, solvable in polynomial time (in terms of *our problem*'s input size, not Knapsack's). In this case, sensors are simply the units in which knapsack weights are expressed, with the knapsack budget equal to n, so the Knapsack DP runs in time $O(mn)$.

In general, though, we will have $e_{ij} = 0$ for some i, j. We can solve the problem by a modification of the standard 0/1 Knapsack DP [95]. We construct an $m \times n$ table of optimal solution values, where $v(j, i)$ is the optimal value of a sub-instance comprising

the first j missions and the first i sensors. Row 1 and Column 1 are initialized to all zeros (optimal profits with the zero-length prefix of missions, and the zero-length prefix of sensors, respectively). We fill in the entries row-by-row, using a value d'_j defined below:

$$v(j,i) = \begin{cases} v(j-1,i), \text{if } M_j \text{ is not satisfiable with } S_1, ..., S_i \\ \max(v(j-1,i),\ p_j + v(j-1, i-d'_j)), \text{o.w.} \end{cases}$$

The two entries of the *max* correspond to satisfying M_j or not. The first option is clear. If we do satisfy M_j in a solution with only sensors $1, ..., i$ available, the important observation is that we may as well use for M_j the rightmost d_j sensors possible, since in this sub-instance M_j is the rightmost mission. (The DP will consider satisfying M_j when solving sub-instances with fewer sensors.) Note that we might be unable to assign the rightmost d_j sensors, since some of these may be too far to the right for M_j to be within their sensing range. Let d'_j equal d_j plus the number of sensors among $S_1, ..., S_i$ too far to the right to apply to M_j. Any sensor too far to the right to reach M_j is also too far to the right to reach M_k for $k < j$. What remains among this sub-instance will therefore be missions $1, ..., j-1$ and sensors $1, ..., i - d'_j$. This completes the recursion definition.

Proposition 7.6.1. *With unbounded budgets, the 1-d problem can be solved optimally in time* $O(mn)$.

Proof. Optimality is by induction. The running time results from the table size and the observation that the budget can be at most n, the number of sensors. □

In the formulation of the infinite-r setting above, the number of sensors n becomes the knapsack budget, whereas in the general setting the sensors must be assigned to missions in range. We now introduce an explicit budget into the problem. Let the model be the same as above, except now also given is an integral budget $B \leq n$, which is the total number

Algorithm 6 1-d 0/1 Dynamic Program

```
1: v(j, i, 0) ← 0 for 0 ≤ j ≤ m, 0 ≤ i ≤ n
2: for b = 1 to B do
3:    v(0, i, b) ← 0 for 0 ≤ i ≤ n
4:    for j = 0 to m do
5:       for i = 0 to n do
6:          if M_j is not satisfiable with b sensors among 1..i then
7:             v(j, i, b) ← v(j − 1, i, b)
8:          else
9:             v(j − 1, i, b) ← max(v(j − 1, i, b),
               p_j + v(j − 1, i − d'_j, b − d_j))
10:         end if
11:      end for
12:   end for
13: end for
```

of sensors that may be used. First delete any missions with $d_j > B$. Then the budgeted version is solved by creating a 3-d DP table, whose third coordinate is budget. We initialize this table by filling in all nm entries for budget 0 with value 0. Then for layer b, we again fill in the entries row-by-row.

Notice that the remaining budget to be spent on the "prefix" instance is $b-d_j$, not $b-d'_j$, since only d_j sensors are actually assigned to mission M_j. Thus we conclude:

Proposition 7.6.2. *Algorithm 3 is optimal and runs in time* $O(mn^2)$.

Proof. Optimality is by induction. The running time results from the observation that the budget can be at most n, the number of sensors. □

We note that the algorithm does not work in the so-called 1.5-d settings, in which the sensors *or* the missions but not both are restricted to a 1-d line. For both cases, counterexamples can be constructed to show that a rightmost mission should not necessarily take

the rightmost available sensors. In 2-d, the 0/1 problem is NP-hard by the reduction from unit-disk MIS [12].

Chapter 8

Sensor Coverage

An ad-hoc sensor network is composed of sensing devices which can measure or detect features of their environment, communicate with one other and possibly with other devices that perform data fusion. One of the problems motivated by ad-hoc sensor networks is to position sensors in order to maximize coverage, or equivalently to minimize the number of sensors required to cover a given area. In such problems, e.g. in monitoring applications, sensors must be positioned so that every point in the region of interest is observable by at least one sensor. In the boolean model of coverage, a sensor is able to observe all points within a certain distance r of its location, i.e., a disk of radius r. The problem is then where to place sensors so that disks centered on them will cover the area. More generally, we may require k-coverage, in which each point must be covered by at least k sensors. This may be required for robustness reasons, in order to perform triangulation, or to obtain multi-modal data. Even in applications requiring only 1-coverage, it may be desirable for power-management reasons to deploy multiple lattices' worth of sensors and then alternate the sensors between active and dormant states (duty cycling [67]), since wireless sensors are typically powered by finite-lifetime batteries.

8.1 Coverage by Lattice

For the 1-coverage problem, placing homogeneous sensors in a hexagon grid pattern (see Fig. 9.1(a)) is known to be optimal [81]: every point is covered at least once and no other covering structure can achieve this property with fewer sensors. Notwithstanding this optimality, there are aspects of this configuration that could be regarded as deficiencies. Since the sensors' sensing regions are circular and therefore not tileable, there inevitably are regions (lens-shaped areas in Fig. 9.1(a), totalling about 20.9% of the total area) that are superfluously covered by two sensors. Second, the configuration is very sensitive. *Any* movement of one sensor from its designated grid position leaves some region uncovered. Third, any sensor failure will similarly cause a gap in coverage.

In the optimal hexagonal covering, the approximately 20.9% of overlap offers no benefit in terms of providing 1-coverage. When the goal is multi-coverage, however, it may be possible to exploit this inherent redundancy. We show that for many values of k, shrinking to increase the number of sensors by a factor of k (often corresponding to combining k separate 1-covers as above) yields k'-coverage for some $k' > k$. We also show that the efficiency ratio k'/k approaches the maximal ≈ 1.209 as $k \to \infty$, e.g., $12/11 \approx 1.09$, $24/21 \approx 1.14$, $58/49 \approx 1.18$. In this sense, the $(k'-k)/k$ ratio is an efficiency gain over the standard hexagon lattice. Perhaps counterintuitively, we provide similar results for the any regular lattice, such as the square lattice in which all of its $\approx 57\%$ overlap is recovered, with efficiency ratios $4/3$ and $7/5$ for small k. This suggests a sense in which the square lattice *is* competitive with the hexagon lattice. For some multi-coverage requirements, i.e., a fixed area and a desired k', a shrunk square arrangement will actually use fewer sensors than the corresponding shrunk hexagon arrangement.

We also consider other benefits of the shrunk lattice arrangement. First, it enjoys greater

robustness against placement error, which has practical advantages. By shrinking the lattice, the sensors will still be deployed according to a hexagonal structure, but with a denser structure allowing the sensors some freedom of movement, without invalidating the full coverage guarantee. Another benefit of shrinking concerns sensor reliability. Depending on sensor type, we may expect a certain percentage of deployed sensors to fail. By sufficiently shrinking the lattice (corresponding to k times as many sensors), we can provide full coverage (in expectation) in the face of a nonzero failure probability, i.e., fault tolerance, and even "spend" the $k \to k'$ efficiency gain on an allowable failure probability.

8.2 Sensing and Wiggle Radii

In sensor networks, a disk is typically used to model the coverage range of a sensor (see e.g. [25]). In the binary coverage model, a sensor's coverage region is a disk centered on the sensor location. There are a variety of coverage problems which solve for optimal locations for sensor placement, or conversely, given a chosen set of sensor locations, solve for optimal (disk-shaped) coverage ranges, according to some objective function (see e.g. [8]). The problem prompting this section is that of assigning radii to sensor points to cover a discrete set of client points, while optimizing some objective function, typically minimizing the sum of the radii $\sum_i r_i$. It is ordinarily assumed that sensors can be (or have been) placed exactly in their *intended* positions. Since sensors are physical objects, however, this assumption of exact placement is only an ideal, which may in realistic situations be violated to various degrees. There may be physical obstructions preventing placement in particular locations, in both urban environments and in nature. More fundamentally, the placement of any physical object is necessarily performed with some amount of error. This fact yields a complementary class of problems in which we seek to maximize the allowance

for placement error, or *wiggle room*, among the sensors. Given a set of positioned unit disks, a dual problem is that of maximizing the sum of the *wiggle radii* $\sum_i w_i$, where w_i is the amount of (directionless) placement error granted to sensor i.

We consider dual classes of geometric coverage problems, in which disks, corresponding to coverage regions of sensors, are used to cover a region or set of points in the plane. The first class of problems involve assigning radii to already-positioned sensors (*being cheap*). The second class of problems are motivated by the fact that the sensors may, because of practical difficulties, be positioned with only *approximate* accuracy (*being flexible*). This changes the character of some coverage problems that solve for optimal disk positions or disk sizes, ordinarily assuming the disks can be placed precisely in their chosen positions, and motivates new problems. We relate the placement problem for unit disks with fixed uniform wiggle radii for guaranteed coverage to coverage solutions for boolean disks. Given a set of disk sensor locations, we show for most settings how to assign either (near-)optimal radius values or allowable amounts of placement error. Our primary results are 1) in the *1-d* setting we give a faster dynamic programming algorithm for the (linear) sensor radius problem; and 2) we find a *max-min fair* set of radii for the *2-d* continuous problems in polynomial time. We also give results for various other settings.

8.3 Focus

Finally, we introduce a geometric coverage problem inspired by an application involving cameras and targets. In the core geometric problem setting, cameras deployed to observe targets in the field may both swivel and adjust their focal length in order to observe one or more targets. That is, for each camera we choose an orientation and a zoom factor. By enlarging a camera's focal length (and thus enlarging its field of view), we allow the camera

to observe more target locations, at the cost of correspondingly reducing the quality of the resulting image; conversely, by zooming in on a small area, the camera will record better quality images of the targets therein, while sacrificing targets lying outside this area. (Ideally, these targets will be covered by other cameras.) The objective, informally speaking, is then configure the cameras in order to observe as many targets as possible, with the highest possible precision.

A camera's configuration consists of its orientation and its zoom factor or field of view (its position is given); the quality of a target's reading by a camera depends (inversely) on both the distance and field of view. We show that the ADD variant (an easy setting in which a target accumulates measurement quality from all cameras observing it) and a geometrically constrained version of the MAX variant are both optimally solvable in polynomial time. We then move on to a more challenging setting in which for each target only the best measurement of it is counted. Although both variants admit continuous solutions, we observe that we may restrict our attention to solutions based on *pinned cones*.

For a geometrically constrained setting, we give an optimal dynamic programming algorithm. For the unconstrained setting of this problem, we prove NP-hardness, present efficient centralized and distributed 2-approximation algorithms, and observe that a PTAS exists under certain assumptions.

For a synchronized distributed setting, we give a 2-approximation protocol and a $(2\beta)/(1-\alpha)$-approximation protocol (for all $0 \leq \alpha \leq 1$ and $\beta \geq 1$) with the stability feature that no target's camera assignment changes more than $\log_\beta(m/\alpha)$ times. We also discuss the running times of the algorithms and study the speed-ups that are possible in certain situations.

This part presents results previously appearing in [66, 13].

Chapter 9

Dense Sensor Deployment

9.1 Introduction

An ad-hoc sensor network is composed of sensing devices which can measure or detect features of their environment, communicate with one other and possibly with other devices that perform data fusion. One of the problems motivated by ad-hoc sensor networks is to position sensors in order to maximize coverage, or equivalently to minimize the number of sensors required to cover a given area. In such problems, e.g. in monitoring applications, sensors must be positioned so that every point in the region of interest is observable by at least one sensor. In the boolean model of coverage, a sensor is able to observe all points within a certain distance r of its location, i.e., a disk of radius r. The problem is then where to place sensors so that disks centered on them will cover the area. More generally, we may require k-coverage, in which each point must be covered by at least k sensors. This may be required for robustness reasons, in order to perform triangulation, or to obtain multi-modal data. Even in applications requiring only 1-coverage, it may be desirable for power-management reasons to deploy multiple lattices' worth of sensors and then alternate

the sensors between active and dormant states (duty cycling [67]), since wireless sensors are typically powered by finite-lifetime batteries.

For the 1-coverage problem, placing homogeneous sensors in a hexagon grid pattern (see Fig. 9.1(a)) is known to be optimal [81]: every point is covered at least once and no other covering structure can achieve this property with fewer sensors. Notwithstanding this optimality, there are aspects of this configuration that could be regarded as deficiencies. Since the sensors' sensing regions are circular and therefore not tileable, there inevitably are regions (lens-shaped areas in Fig. 9.1(a), totalling about 20.9% of the total area) that are superfluously covered by two sensors. Second, the configuration is very sensitive. *Any* movement of one sensor from its designated grid position leaves some region uncovered. Third, any sensor failure will similarly cause a gap in coverage.

A 2-covering obtained by overlaying two translated copies of the optimal 1-covering is shown in Fig. 9.1(b). We give similar coverings for $k > 2$. For robustness reasons, we obtain a k-covering by arranging k translated copies of the hexagonal lattice to maximize the minimum pairwise distance d_{min} between sensors. We show that this reduces to *distributing k points evenly on the torus*. This problem can be approached by *minimizing a Riesz s-energy* objective function with a sufficiently large parameter s. We show examples in which the s value needed varies based on the value of k. It turns out that for larger values of k, the solution provided by optimizing Riesz energy is often simply a denser hexagon lattice, or is close to this. The second method we consider is therefore simply shrinking the lattice, i.e., deploying sensors in a more densely packed hexagon arrangement, thus avoiding the optimization calculation.

In the optimal hexagonal covering, the approximately 20.9% of overlap offers no benefit in terms of providing 1-coverage. When the goal is multi-coverage, however, it may be

possible to exploit this inherent redundancy. We show that for many values of k, shrinking to increase the number of sensors by a factor of k (often corresponding to combining k separate 1-covers as above) yields k'-coverage for some $k' > k$. We also show that the efficiency ratio k'/k approaches the maximal ≈ 1.209 as $k \to \infty$, e.g., $12/11 \approx 1.09$, $24/21 \approx 1.14$, $58/49 \approx 1.18$. In this sense, the $(k' - k)/k$ ratio is an efficiency gain over the standard hexagon lattice. Perhaps counterintuitively, we provide similar results for the any regular lattice, such as the square lattice in which all of its $\approx 57\%$ overlap is recovered, with efficiency ratios $4/3$ and $7/5$ for small k. This suggests a sense in which the square lattice *is* competitive with the hexagon lattice. For some multi-coverage requirements, i.e., a fixed area and a desired k', a shrunk square arrangement will actually use fewer sensors than the corresponding shrunk hexagon arrangement.

For example, in the case of $k = 5$, we obtain $k' = 7$, which yields $k'/k = 7/5 = 1.4$ and $1.57/(k'/k) \approx 1.12$, which is better than the efficiency 1.209 of the shrunken hexagon lattice.

We also consider other benefits of the shrunk lattice arrangement. First, it enjoys greater robustness against placement error, which has practical advantages. If the designated position for the sensor is not ideal, because of difficult terrain, for example, or because of man-made obstructions, it is natural to place the sensor nearby and hope to continue to provide full coverage. Or, if (some of) the sensors are mobile, they might take advantage of any available *wiggle room* to move about and avoid detection. By shrinking the lattice, the sensors will still be deployed according to a hexagonal structure, but with a denser structure allowing the sensors some freedom of movement, without invalidating the full coverage guarantee. We call the area in which a sensor is free to move its *wiggle area* and explore its shape. We note that this area need not be a disk, though the maximum *wiggle*

disk[1], corresponding to a sensor's maximum allowable displacement in any direction, will be of particular practical interest. Conversely, we compute the lattice density (degree of shrinking) required in order to support a certain desired amount of allowable placement error.

Another benefit of shrinking concerns sensor reliability. Depending on sensor type, we may expect a certain percentage of deployed sensors to fail. Alternatively, in event detection applications, there may be a known false negative probability. These considerations provide a third kind of benefit to shrinking the sensor network. By sufficiently shrinking the lattice (corresponding to k times as many sensors), we can provide full coverage (in expectation) in the face of a nonzero failure probability, i.e., fault tolerance, and even "spend" the $k \to k'$ efficiency gain on an allowable failure probability.

Model. We assume a boolean sensor coverage model, with sensing coverage areas as disks. While this is a simplified model, it can be used to provide conservative bounds in applications with more complex anisotropic or probabilistic sensor models [11], and is interesting enough to be widely studied on its own. Throughout the paper, we assume a large convex area of interest, which allows us to ignore the edges. If connectivity is also required, then we assume that $r_c \geq 2r_s$, i.e., the communication range is at least double the sensing range; this is known to suffice for inferring that full coverage implies connectivity [104], and so we do not further discuss connectivity in this paper.

Organization. In Sect. 11.2 we present related work. In Sect. 9.3 we explore a Riesz energy technique for finding robust k-coverage arrangements before turning to the simpler method of simply shrinking the lattice. We demonstrate the efficiency of the shrunk lattice for multi-covering, showing that the equivalent of k copies can provide k'-coverage. We

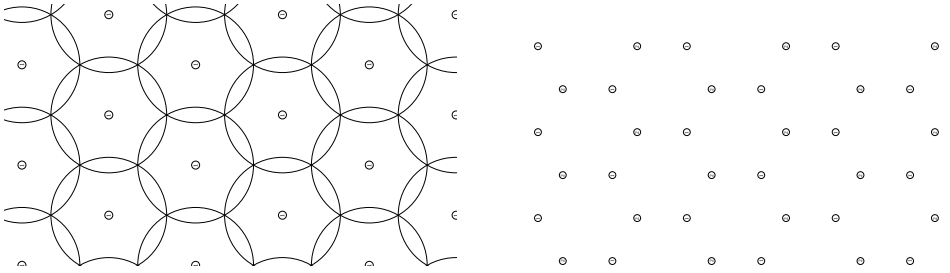
[1] *Defined as the largest disk centered at the sensor that is contained within the wiggle area.*

give a series of examples, converging in the limit, in which the percentage of recovered area approaches the $\approx 20.9\%$ maximum. Second, we quantify the flexibility benefits of shrinking the hexagon lattice, characterize the resulting *wiggle area*, prescribe the amount of shrinking required to support a given flexibility requirement (Sect. 9.4). Third, we study the fault-tolerance provided by shrinking (Sect. 9.5). We conclude by construing the shrinking factor as a budget divided between these benefits (Sect. 9.6).

9.2 Related Work

There have been efforts in various communities to characterize, find approximate solutions to, and analyze the intrinsic properties of, the coverage problem based on the boolean model. [25] focused on theoretical bounds for various coverage requirements using static as well as mobile sensors, all under the boolean model. [104] gave a distributed algorithm that attempts to minimize the number of sensors needed to obtain full coverage, by minimizing the amount of intersection area of sensor disks. [50] gave an $O(\log n)$ Set Cover-style greedy algorithm for the setting in which sensor locations must be chosen for a given discrete set, which was extended to k-coverage in [107]. [42] proved a matching $\Omega(\log n)$ approximation ratio lower bound. The setting in which only a discrete set of clients must be covered is also well studied [8]. In this paper, however, our focus is on the setting of full coverage with unrestricted sensor placement.

Some research combines coverage goals with connectivity or other constraints. [11], for example, provided sensor deployments yielding 1-coverage and either 1-connectivity or 2-connectivity. Potential field methods have been used to distribute sensors so that in the resulting graphs sensor nodes have a minimum degree [85].



(a) A rigid 6-grid (hexagon lattice). (b) A 2-covering of the plane.

Figure 9.1: Two grid arrangements.

[13] discussed some other problems related to inexact sensor placement. A connection was also drawn between probabilistic placement error and probabilistic (non-binary) sensing models. [108] combined probabilistic placement error and probabilistic sensing models in seeking a high-confidence full cover of grid points. They presented experimental results of a heuristic algorithm that greedily selects grid points with low current coverage probability.

There is a large literature on k-coverage. [99] provided k-coverage arrangements, for binary and probabilistic sensors. They noted the presence of overlap among disks but take advantage of it only in the setting of small r_c. For large r_c, which is our focus here, they simply duplicate the 1-covering k times. [17] gave distributed algorithms for assembling sensors into a hexagon lattice, whereas we take the ability to position sensors as a given and study the benefits of such arrangements. Focussing on probabilistic sensors, [103] gave bounds on density for guaranteeing *full k-coverage with high probability* with various arrangements. This non-local guarantee requires increasing sensor density by a logarithmic factor. Our focus in Sect. 9.5, however, is purchasing a local coverage guarantee, using only a constant factor increase in density. [58], for example, gave efficient algorithms for determining the maximum value k for which a given sensor configuration provides

k-coverage, supporting both unit-disk sensors and non-unit-disk sensors.

Finally, [82] studied the problem of decomposing a k-cover into a maximum number of disjoint 1-covers. They noted that when the region of interest is a disk, the only positive result that has been claimed is that any 33-cover may be decomposed into two disjoint 1-covers. [6] solved a related but relaxed problem in which a set of sensors is partitioned into subsets with the objective of maximizing the sum of the (partial) covers of the sets. We remark that our second motivation for shrinking the lattice is the reverse of this, *combining* multiple 1-covers.

9.3 k-coverage

Under k-coverage, each point in the region is covered by at least k different sensors. A 2-covering (see Fig. 9.1(b)) can be obtained by concatenating two optimal 1-coverings where there is a translation between the two 1-coverings. This approach enjoys a robustness property, viz., the sensors are placed maximally far away from one another. We now explore how k-coverings for higher k can be achieved that also satisfy this property. Sufficiently increasing the density of the sensor network lattice will yield a k-fold coverage. Furthermore, we will see how to exploit the overlap in a single lattice and improve the efficiency of the obtained coverage relative to the number of sensors used.

9.3.1 Riesz Energy

We construct a k-covering by superimposing k translated copies of the hexagonal 1-covering. Because of the lattice structure of the optimal 1-covering, each sensor within the 1-covering can be construed as lying on the surface of a torus, i.e., the 1-covering is reduced to a single

sensor located on the surface of a torus. Since this lattice is the same for all the 1-coverings, the tori for the different 1-coverings can be thought of as the same object. The distance metric d on the torus is induced by the distance metric d_R in the region R. Since each point on the torus corresponds to a lattice in the plane, when unwrapped onto R, d is the Hausdorff distance between two lattices. In this case, for robustness reasons, i.e. to maximize $d_{\min} = \min_{1 \le i < j \le k} d(a_i, a_j)$, the problem is reduced to dispersing k points on a torus[2]. This problem has been studied using the approach of Riesz energy minimization [51]. The s-Riesz energy of a set of points $\{a_i\}$ is defined as

$$E_s = \sum_{i \neq j} \frac{1}{d(a_i, a_j)^s}$$

for a distance metric d and exponent $s > 0$. For $s = 0$, the Riesz energy is defined as

$$E_0 = \sum_{i \neq j} \log \frac{1}{d(a_i, a_j)}$$

By minimizing E_s for $s \to \infty$ using optimization algorithms, a distribution of points is found which appears to solve the best packing problem, i.e., maximizing $d_{\min}$. Quasi-Newton based optimization algorithms have been developed for cases where the number of points is in the thousands [100].

9.3.2 Experimental Results

By minimizing the Riesz energy E_s, we obtain configurations of k-coverings for various values of k. In our experiments, we choose $s = 10$ and $d_R(a_i, a_j) = \|a_i - a_j\|_2$ to be the Euclidean distance on the plane. For 2-coverings, we obtain the same result as shown

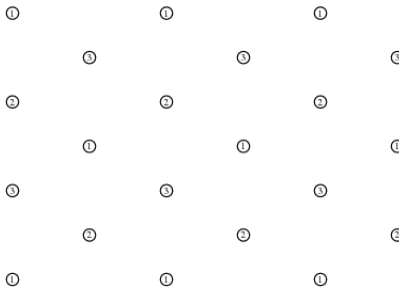
[2]The problem of maximizing the pairwise distance between a fixed number of points has previously been studied on the sphere, where it is known as Fejes Tóth's problem [26]. Exact solutions are known for $k \le 12$ and $k = 24$, but no general solutions are known.

in Fig. 9.1(b). The results for k-coverings for various values of $k > 2$ are shown in Fig. 9.2. In these figures, sensors labeled with the same number belong to the same hexagonal 1-covering. Note that the union of all the sensors can form a regular geometric structure, and in some instances a hexagonal lattice. For instance, for $k = 3, 4, 7, 9, ...$ (see Figs. 9.2(a),(b) and 9.3(b)), the union of all the sensors form a hexagonal lattice, the optimal 1-covering. More generally, let Q be a set of of integers q such that a hexagonal 1-covering can be partitioned into a q-covering after rescaling the distances by a factor of $\sqrt{q}$. By recursively partitioning this way (and scaling the sensor locations appropriately), it can be seen that that for k-coverings, with k of the form $k = \Pi_{q_i \in Q} q_i^{m_i}$, $m_i \in \mathbb{Z}_0^+$, there is an optimal configuration where the union of all the sensors form a hexagonal lattice. In this case, $d_{min} = \frac{1}{\sqrt{k}}$. For any k', if $k = \Pi_{q_i \in Q} q_i^{m_i}$ is the smallest integer of this form such that $k' \leq k$, then a subset of the optimal k-covering described above can serve as a k'-covering and $\frac{1}{\sqrt{k}}$ is a lower bound for the best $d_{\min}$ for a k'-covering. On the other hand, because of the optimality of the hexagonal lattice, $\frac{1}{\sqrt{k'}}$ is an upper bound for $d_{\min}$ for a k'-covering.

Similarly, given a k'-covering, we can extend it to a $k'\Pi_{q_i \in Q} q_i^{m_i}$-covering by recursively replacing 1-coverings with optimal q_i-coverings. In general, though, this covering is suboptimal.

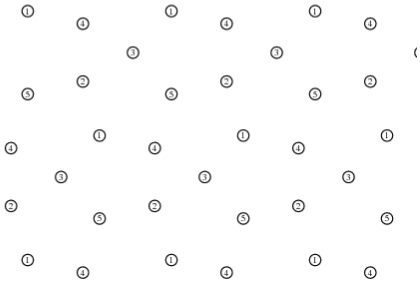
Let Q' be the set of all integers k' for which there is an optimal k'-covering such that the union of sensors form a hexagonal lattice. Computing Q' is equivalent to determining the indices for which there is a sublattice of a hexagonal lattice which is also a hexagonal lattice. It was shown in [20] that $Q' = \{a^2+ab+b^2 | a, b \in \mathbb{Z}^+\}$. Another way to characterize Q' is that for $k' \in Q'$, all prime factors of k' of the form $3m + 2$ have even exponents[3]. We choose the set Q as the maximal subset of Q' such that each element in Q is not a product

[3] http://www.research.att.com/~njas/sequences/A003136

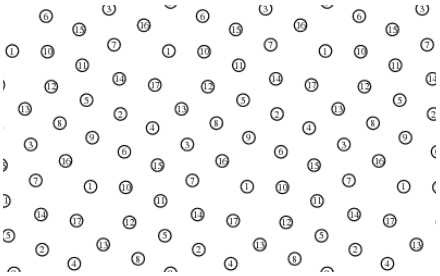


(a) A 3-covering of the plane with $d_{\min} = \frac{1}{\sqrt{3}}$. Note that the union of 3 hexagonal 1-coverings forms a hexagonal covering.

(b) A 4-covering of the plane with $d_{\min} = \frac{1}{2}$. Note that the union of the 4 hexagonal 1-coverings is again a hexagonal covering.



(c) A 5-covering of the plane with $d_{\min} = 0.3937$.



(d) A 17-covering of the plane with $d_{\min} = 0.2198$.

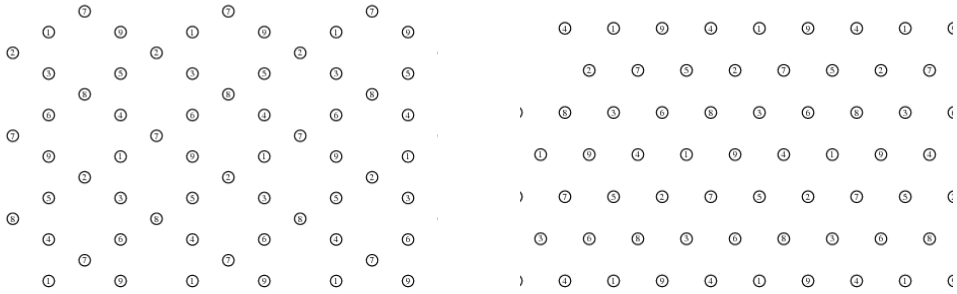
Figure 9.2: Computed k-coverings for various values of k.

of elements of Q and in this case $Q = \{3, 4, 7, 13, 19, 25, ...\}$ is the set of the norms of Eisenstein-Jacobi primes[4] and $Q' = \{\Pi_{q_i \in Q, m_i \geq 0} \quad q_i^{m_i}\}$.

Choosing the Exponent s

How large should the value of the exponent s be when computing the Riesz energy E_s? In order to approximate the energy function that solves the best packing problem, s should be chosen to be large. However, a large s leads to large gradients and can cause problems in the numerical computations. On the other hand, in our experiments we found that choosing

[4] `http://www.research.att.com/~njas/sequences/A055664`



(a) Optimized using $s = 1$. $d_{\min} = \frac{1}{2\sqrt{3}}$. (b) Optimized using $s = 2$. $d_{\min} = \frac{1}{3}$.

Figure 9.3: Two 9-coverings of the plane.

small values of s can produce suboptimal results. For instance, we show in the subfigures of Fig. 9.3 the 9-coverings produced by optimized Riesz energy with $s = 1$ and $s = 2$, respectively. According to the discussion above, the latter subfigure illustrates the optimal configuration maximizing d_{min}. Numerical experiments show that there is a transition from one configuration to the other configuration around $s = 1.2$.

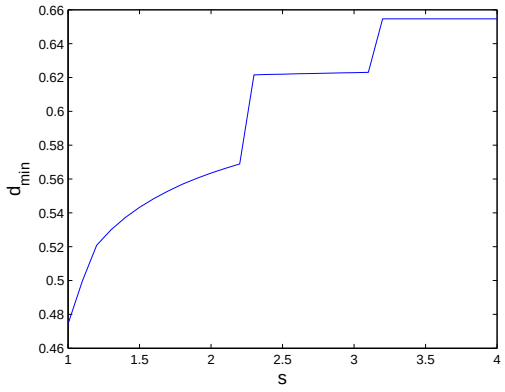


Figure 9.4: $d_{\min}$ obtained in a 7-covering by minimizing E_s for various s.

In fact, by the so-called Poppy-seed Bagel Theorem [51], as the number of points k approaches infinity, minimizing the s-energy for *any* value $s \geq 2$ will distribute the points uniformly on the surface of the torus, thus maximizing $d_{\min}$. This result only refers to k in the limit, however, and may not hold for small k. For small values of k, exponent s

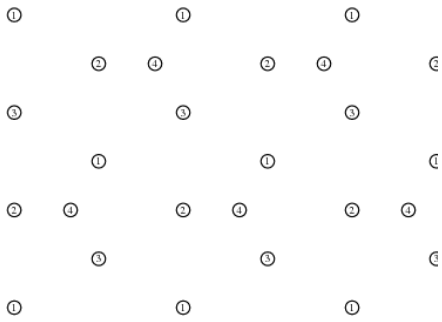
might need to be large in order to maximize $d_{\min}$. In practice, it could be difficult to know for certain whether the s value chosen is large enough for the chosen k. For example, for $k = 7$, Fig. 9.4 shows $d_{\min}$ as E_s is minimized for various values of s. We see that $d_{\min}$ reaches the optimal value of $\sqrt{3}/\sqrt{7}$ at $s \approx 3.2$, after passing through several other configurations. On the other hand, for $k = 9$ the only transition occurs at $s \approx 1.2$. As we noted before, we cannot simply increase s until we obtain a hexagonal solution since the optimal configuration need not be hexagonal.

9.3.3 Incremental k-coverings and Other Base Coverings

Consider the scenario where a k-covering is in place and we want to expand the sensor network to a k'-covering ($k' > k$) by adding more sensors and the question is where the new sensors should go. Again the same optimization approach can be used, with the additional constraint that the sensors in the original k-covering remain fixed. In Fig. 9.5(a) we show a 4-covering obtained by extending the 3-covering in Fig. 9.2(a). Compare this with the optimal 4-covering in Fig. 9.2(b).

So far we have applied this algorithm to the case where the underlying 1-covering (the *base* covering) is the hexagonal lattice. This approach can also be applied to other base 1-coverings. For instance, Fig. 9.5(b) shows a 4-covering obtained by translating 4 square grid 1-coverings and optimizing the arrangement by minimizing E_{10}. We see that the totality of sensors form a hexagonal lattice. A more irregular looking 6-covering based on 6 squared grid lattices is shown in Fig. 9.6(a).

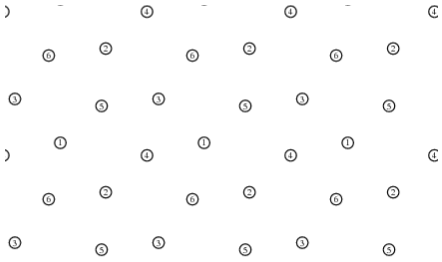
There is a close relationship between these k-coverings and dispersed dither digital halftoning [94]. In dispersed dither, a pattern is tiled on the plane such that it forms a visually pleasing pattern. The dispersed ordered dither patterns (e.g., blue noise dispersed



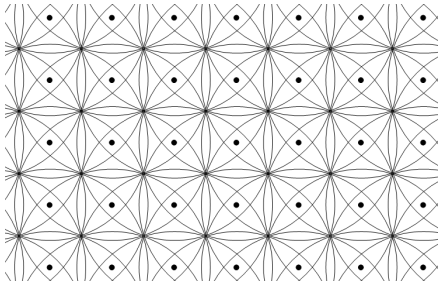
(a) $d_{\min} = \frac{1}{3}$. This covering is obtained by adding an additional 1-covering to the 3-covering in Fig. 9.2(a).

(b) The base 1-covering is the square grid lattice and $d_{\min} = \frac{1}{2\sqrt{2}}$.

Figure 9.5: Two 4-coverings of the plane.



(a) A 6-covering of the plane. The base 1-covering is the square grid lattice and $d_{\min} = 0.2708$.



(b) The square lattice shrunk by factor 5, providing 7-covering.

Figure 9.6: Two grid arrangements.

dither [92]) are regular patterns that are repeated and similar to k-coverings such as Fig. 9.6(a). The stacking constraint in these dither patterns dictates that a pattern with more dots is a superset of a pattern with fewer dots, the same constraint as with the incremental k-coverings discussed above. Indeed, many algorithms for generating blue noise dither patterns are based on energy minimization [101]. In a k-covering, however, the sensors can be positioned at any point on the plane resulting in a linear programming problem, whereas in a dispersed dither pattern, the dots are constrained to lie on the printer addressability grid (e.g. a 600 dpi or a 1200 dpi grid), resulting in an integer programming problem.

9.3.4 Analysis

The observation that some k-coverings generated by this method have all the sensors themselves form a denser hexagon lattice suggests another scheme for obtaining k coverage: take the standard hexagon lattice, initially drawn on a scale where $d_{min} = \sqrt{3}$ and $r = 1$, and *rather than minimizing Riesz energy*, simply increase its density (or equivalently increase the value r) until k-coverage has been achieved. Notice that there is an equivalence between increasing the density and increasing r: we always have $g = r^2$, where r is the sensor radius and g is the number of *hexagon lattices' worth of sensors* used. For example, in the unscaled hexagon lattice we have $g = 1 = r^2$.

Unlike in single coverage, a hexagon lattice is not the most efficient configuration for every value k, e.g. consider the 2-coverage configuration shown in Fig. 9.1(b). By what factor must the hexagon lattice be shrunk in order to provide, e.g., 2-coverage? Consider a neighborhood $N_\epsilon(x)$ centered on point x which is a sensor location in the hexagon lattice. The first cover of this area comes from the sensor located at x. The second cover must be provided by (perhaps some of) the sensors surrounding it, i.e., x's six neighbors in the hexagon lattice. Unfortunately, any radius assigned to those points that yields an additional cover for $N_\epsilon(x)$ will actually provide *two* more covers for it. That is, the required shrinking factor corresponds to three times as many sensors as a single lattice, i.e., the hexagon lattice with $g = 3$ or $r = \sqrt{3}$. The corresponding arrangements of disks is known as the Flower of Life. Naively shrinking until we double the number of sensors (i.e., until $g = 2$) would fail to produce 2-coverage. Riesz energy minimization provides a 2-cover, using two copies of the hexagon lattice, with $d_{min} = r = 1$ (see Fig. 9.1(b)). Nonetheless, for many values (including but not limited to $k \in Q'$) shrinking performs well.

The proposed k-covering satisfies a stronger property of providing k-coverage using

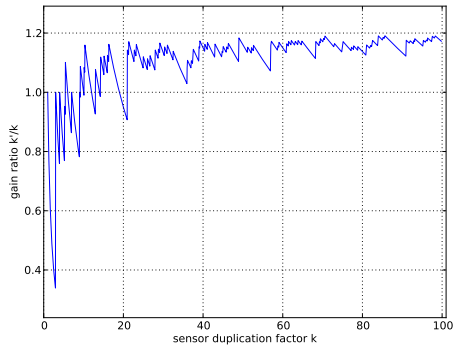
sensing range	k	k'	gain ratio
3.31662	11	12	1.09091
3.60555	13	14	1.07692
3.87298	15	16	1.06667
4.00000	16	18	1.12500
4.12311	17	19	1.11765
4.24264	18	19	1.05556
4.58258	21	24	1.14286
4.69042	22	25	1.13636
4.79583	23	26	1.13043
4.89898	24	27	1.12500
5.00000	25	28	1.12000
5.09902	26	28	1.07692
5.19615	27	30	1.11111

Table 9.1: $k \to k'$ values for $k \in [1, 27]$ s.t. $\frac{k'}{k} > 1$.

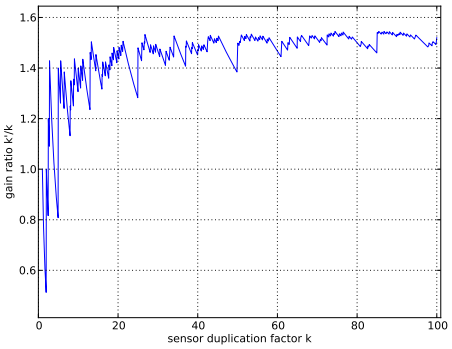
one sensor from each of the underlying k lattices, which may be useful in multi-modal data fusion applications, where there are k types of sensors and each point in the region of interest needs to be covered by at least one sensor of each type. On the other hand, there is overlap in the coverage area, i.e. there are areas in a 1-covering that are covered by more than 1 sensor (e.g. Fig. 9.1(a)). We will show in Section 9.4 that the overlap density ρ is equal to $\frac{\pi r_c^2}{|ad-bc|}$, which is $\frac{2\pi}{3\sqrt{3}} \approx 1.209$ and $\frac{\pi}{2} \approx 1.57$, for the hexagonal and square lattices, respectively.

In the absence of such data fusion applications, we can take advantage of this overlap, i.e. for sufficiently large k, the k-covering constructed is actually a k'-covering, with $k'/k \approx \rho$.

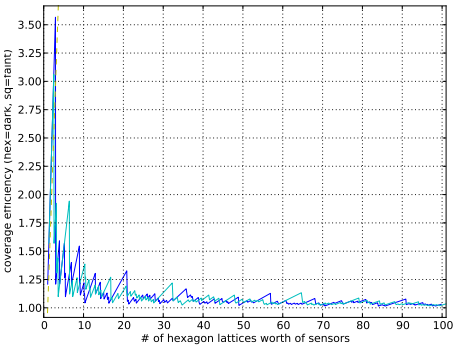
Proposition 9.3.1. *For all $\epsilon > 0$ there exists k such that the k-covering constructed by concatenating k 1-coverings is also a k'-covering, and $k'/k > \rho - \epsilon$.*



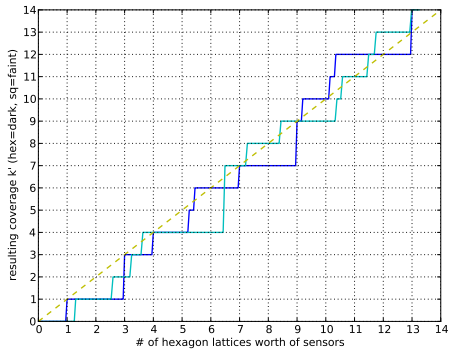
(a) Shrunken hexagon lattice efficiency gain $\frac{k'}{k}$ v. k.



(b) Shrunken square lattice efficiency gain $\frac{k'}{k}$ v. k.



(c) Coverage efficiency (coverage size / sum of disk sizes)



(d) Coverage provided versus cost

Figure 9.7: Computational results.

Proof: Let k be chosen such that the lattice can be decomposed as k identical sublattices, e.g. we pick $k \in Q'$ for the hexagonal lattice. The distance between the sensors is $1/\sqrt{k}$ times the corresponding distance in the sublattice. For each point in the region consider a circular region of radius r_c. The number of lattice points in this region is approximately $k \cdot \frac{\pi r_c^2}{|ad-bc|}$ as $k \to \infty$.[5] Thus this is a k'-covering where $k' \approx k \cdot \frac{\pi r_c^2}{|ad-bc|} = k\rho$.

In fact, ρ_{opt} upperbounds the maximum achievable gain.

Proposition 9.3.2. *k'/k cannot exceed ρ_{opt}.*

[5]There is a large body of work studying the closeness of this approximation as a function of k, see e.g. [62].

Proof: We consider the hexagon lattice case here, but the same argument is valid for other lattices. Suppose n sensors appear in the hexagon lattice covering a certain area of size A. Suppose nk sensors (i.e., k hexagon lattices' worth) provide a k'-coverage. A hexagon lattice covers the entire area at least once and approximately 20.9% of it twice i.e., $nS \approx 1.209A$, where S is the size of a single sensor's sensing range. Then the total amount of area covered by k lattices' worth of sensors is $knS \approx k1.209A$. Therefore with kn sensors, the best we could hope for is that the full area A is covered $\approx 1.209k$ times, i.e., $k'/k \leq \rho_{opt}$.

We show the following by machine computation. We verified the results by hand for several cases.

Proposition 9.3.3. *For several particular shrinking factors (shown in Table 9.1) corresponding to a k-fold duplication of the hexagon lattice, a k'-coverage, for some integer $k' > k$, is provided. In particular, 11 copies of the hexagon grid suffice to provide 12-coverage.*

Proof: The multiplicity of coverage provided by a given arrangement can be found efficiently. The Huang and Tseng [58] algorithm does so by computing the minimum value k' for which all sensor disks are *k'-perimeter covered*, which means that all points on the perimeter are covered by k' sensors other than itself, on the assumption that no two sensors lie at the same location. Let r be the sensing range. Then computing the perimeter coverage of one sensor s can be done in time $O(c \log c)$, where c is the number of sensors within distance $2r$ of s. Due to symmetry, we only need to compute the perimeter coverage of a single sensor in the hexagon lattice (in fact, for only $1/6$ of its perimeter). We did this for the values k appearing in Table 9.1. Since there are $O(r^2)$ sensors in the hexagon lattice within distance r of any point, the total running time will be $O(r^2 \log r)$.

In Fig. 9.7(a) we show the obtained efficiency gain k'/k for various values of k. We see that it approaches $\rho_{opt} = 1.209$ for large k, as promised by Proposition 9.3.1. Analogously, Fig. 9.7(b) shows the obtained efficiency gain for a shrunken square lattice, which correspondingly approaches a recovery of the $\approx 57\%$ inefficiency of the square grid. We emphasize here that although the square lattice is typically considered inferior to the hexagon lattice, these results argue that the hexagon's advantage disappears when comparing *shrunken* lattices. In the limit, all the single lattice's redundancy is recovered.

Fig. 9.7(c) plots the "coverage efficiency" of the two lattices as we vary the number of sensors used. For both curves in this figure, the X coordinate indicates the number of sensors used, in units of *one hexagon lattice's worth*. The Y coordinate indicates the efficiency, i.e. $\rho_{lat}/(k'/k)$. This efficiency measure is equivalent to coverage area size divided by the total coverage "paid for" (the number of sensors times the coverage area size of one sensor). Following the known discrepancies in this efficiency for values $k = 1$ and $k = 2$, we observe as expected that both efficiency ratios converge to unity in the limit. Finally, Fig. 9.7(d) illustrates a portion of this data differently, plotting the coverage provided by hexagon and square lattices as a function of the number of sensors used (varying from 1 to 14 hexagon lattices' worth). Note that each time one of these curves supersedes the dashed $y = x$ line, it corresponds to a particular shrunken lattice arrangement is more efficient than the base hexagon lattice.

Again we find potential grounds for using the square lattice. Suppose, for example, that we wish to cover a given area with 4-coverage. Shrinking the hexagon lattice by a factor of 4 will provide 4 coverage, yielding the same efficiency ratio as simply using four copies of the hexagon lattice. Shrinking the square lattice by a factor of 3, however, will provide 4-coverage while using $3 \cdot \rho_{sq}/\rho_{hex} \approx 3 \cdot 1.57/1.209 \approx 3.896$ times as many

sensors as one hexagon lattice. Or, if our goal is e.g. 7-coverage (see Fig. 9.6(b)), we find that shrinking the square lattice by a factor of 5 will provide 7-coverage for the "cost" of $\approx 5 \cdot 1.57/1.209 \approx 6.493$ hexagon lattices. That is, the hexagon lattice shrunk by a factor of 7 provides 7 coverage, using approximately $7 \cdot 1.209 \approx 8.463$ sensors per unit area, whereas the square lattice shrunk by a factor of 5 provides the same coverage with a per-unit area sensor count of only about $5 \cdot 1.57 \approx 7.85$.

9.4 Wiggle Room

As we noted above, the optimal hexagon lattice is very rigid in the sense that the movement of a single sensor will result in uncovered space. In this section, we will explore a trade-off between density and position flexibility.

9.4.1 Preliminaries

In the hexagon lattice configuration [26, 81], each sensor disk is overlapped by six others (see Fig. 9.1(a)). The location of the sensors form a grid in which the nearest pair of sensors are $r\sqrt{3}$ apart, where r is the sensing radius. This quantity, which we indicate by d_{min} (or just d when clear from context), is sometimes referred to as the grid's *granularity*. The *coverage density* ρ of an arrangement of sensors is the ratio of the sum of the area of coverage regions for each sensor to the area of the region to be covered. An arrangement minimizing this quantity would typically be preferred. The density ρ_{opt} of that arrangement is simply the ratio of the areas of circle and hexagon:

$$\rho_{opt} = \frac{2\pi}{3\sqrt{3}} \approx 1.209 \tag{9.4.1}$$

Hence the number of sensors n_{opt} needed to cover a region of area A is given by:

$$n_{opt} = \frac{A \cdot \rho_{opt}}{\pi r^2} = \frac{2A}{r^2 3\sqrt{3}} \approx \frac{0.385A}{r^2} \tag{9.4.2}$$

Although this configuration uses fewer sensors than any other, many other grid formations could be considered, some of which have practical advantages. One example is the square lattice, which has the following coverage density and sensor count, respectively:

$$\rho_{sq} = \frac{\pi}{2} \approx 1.57 \tag{9.4.3}$$

$$n_{sq} = \frac{A}{2r^2} \tag{9.4.4}$$

We refer to a lattice in which each node has m neighbors as an m-grid. To calculate the amount of overlap in a general lattice covering, let (a, b) and (c, d) be the two basis vectors of the lattice, e.g. $(a, b) = (\frac{r\sqrt{3}}{2}, \frac{r3}{2})$, $(c, d) = (r\sqrt{3}, 0)$ for the hexagonal lattice and $(a, b) = (r, 0)$, $(c, d) = (0, r)$ for the square lattice. Then the coverage density is equal to $\frac{\pi r^2}{|ad-bc|}$, which generalizes the two densities stated above.

Although our main focus is on the 6-grid, the 4-grid is popular because of practical convenience, and so we sometimes include calculations for the 4-grid for comparison. By a *rigid* m-grid we mean an m-grid with the smallest number of sensors among those m-grid arrangements providing full coverage. An operational difficulty of these rigid m-grid solutions is that sensors must be placed exactly, with no room for placement error. After all, any room for error would imply that the grid was not tight and so used more sensors, or sensors with larger sensing radii, than necessary. Conversely, if we do use more or larger-radius sensors than the absolute minimum needed, there may be flexibility in allowable sensor placement while maintaining full sensing coverage over the region. We refer to the region within which a sensor can be placed while full coverage is maintained as the sensor's

wiggle region. Given a particular feasible configuration (meaning one which covers the entire region of interest), we may be interested in the wiggle regions of all the individual sensors. For sensors in any lattice configuration, the wiggle regions would be congruent but translated; for an arbitrary arrangement, different sensors might have different sorts of wiggle regions. Given a placement of sensors, we may compute the shape and size of the wiggle region for each.

A sensor's *intended location* is the location where it lies if there is zero placement error. A simple and practical way of characterizing the size of a wiggle region is by the *wiggle radius* w, i.e., the radius of the largest circle *centered on the intended sensor location* inscribable in it. We call this circle the *wiggle disk*. The wiggle disk will be of practical importance when placement error is isotropic, i.e., the same for all directions. For some configurations, such as the 6-grid, the difference between the wiggle disk and the full wiggle area will be quite small, though this need not be the case in general.

If (conversely) we are given desired amounts of allowable placement error, we could compute a configuration that is robust enough to continue to guarantee full coverage in spite of this error. While grid-based solutions have desirable properties and are straightforward to apply, it may in reality be impractical to place sensors exactly on grid points. A dual problem is to find an efficient configuration that compensates for the placement error that we wish to allow, and therefore assume to occur. It is for this second problem that we provide the denser hexagon lattice as a solution.

9.4.2 From Arrangements to Wiggle Regions

In this section, we examine the scenario in which a sensor and its neighbors may be inexactly positioned. There must be some overlap among sensors' coverage disks in any

arrangement fully covering a region. Given a sensor, we refer to the region not covered by any other sensor's coverage disk as that sensor's *responsibility region* (RR). We now construct the wiggle region, comprising all those points on which this sensor could be repositioned such that full coverage is maintained, for these two settings.

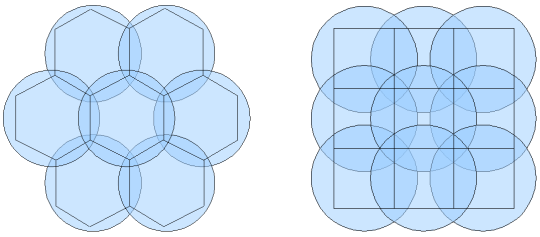


Figure 9.8: In 6-grid and 4-grids, congruent responsibility regions that tile the region.

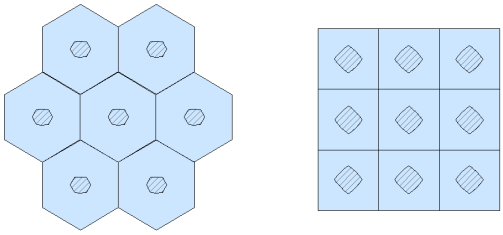


Figure 9.9: In 6-grid and 4-grids, wiggle regions such that sensors cover their responsibility regions.

In this scenario, all the sensors can be repositioned from their intended position, and we seek a characterization of wiggle regions. We focus on grid-based solutions with congruent wiggle regions.

We begin by generalizing the notion of responsibility region for s, i.e., the region that sensor s must fully cover no matter where the other sensors lie, to the setting in which multiple sensors may be placed inexactly. By achieving coverage for each RR, we achieve full coverage. We say that s is responsible for any point p whose distance to the initially designated position of sensor s is no greater than its distance to the initially designated

position of any other sensor, in which case s's responsibility region is simply its Voronoi neighborhood [81]. Fig. 9.8 shows the boundaries of the Voronoi neighborhoods of sensors in grid arrangements. Due to symmetry in grid arrangements (and due to the nature of Voronoi diagrams), these RRs will tile the plane, as shown in Fig. 9.8.

The wiggle regions based on these congruent responsibility regions can once again be constructively obtained. Let R be the bounded responsibility region of a particular sensor s with sensing radius r, and let P be the set of extreme points of R. Then full coverage is maintained if and only if s lies in the intersection of closed disks with radius r centered at the points in P. We take the extreme points to be the Voronoi points in the resulting Voronoi diagram.

Each responsibility region R in a grid-based arrangement is its own convex hull, and will be covered by placing a sensor anywhere within the intersection of radius-r disks centered at R's extreme points. We show the wiggle regions for a 6-grid in Fig. 9.9. Since each sensor can be placed within its wiggle region independently of the choice of where others are placed within their wiggle regions, the placement policies must naturally be more conservative. Treating Voronoi neighborhoods in arbitrary arrangements as responsibility regions implies corresponding wiggle regions as well. Since Voronoi neighborhoods are always convex, the construction in Proposition 1 can be applied in every case in order to obtain the wiggle region.

In a general arrangement of sensors, given a designated sensor with position p, let X be the set of extreme points of its responsibility region, and for each $q \in X$, let $s(q)$ denote the distance between p and q. Let $d(q)$ denote the distance between q and the boundary of the same sensor's coverage disk. That is, $d(q) = r - s(q)$ for $0 \leq s(q) \leq r$. Then the disk centered at p with radius $\min_{q \in X} d(q)$ necessarily fits within the wiggle region. In the field,

it may be easier to work with this smaller disk in lieu of the whole wiggle region, since it is easier to calculate and specify. Note that for non-lattice arrangements, this need not be the largest disk that can be inscribed in the wiggle region.

9.4.3 From Desired Flexibility to Required Density

We use the characterization above in calculating the lattice granularity (and thus the number of sensors) required for covering a region with an m-grid, in accordance with a minimum required wiggle radius.

Suppose we are given an upperbound on placement error w, and we would like to find a grid based arrangement that attains full coverage. Observe that it is necessary and sufficient to position sensors in a m-grid such that a circle of radius w can be inscribed in the wiggle region of every hypothetical sensor placed exactly at grid points. In other words, no matter where the sensor is placed inside this disk, its distance to every extreme point of the (Voronoi) responsibility region must be at most r. The best arrangement, i.e., one with the largest grid granularity, under these additional constraints is one in which the distance from every critical point to the center of a Voronoi region is exactly $r - w$.

By definition, in a rigid m-grid arrangement, the distance from each sensor to the extreme points of its Voronoi region is r; we can allow for wiggle room w by scaling the rigid m-grid by a factor of $(r-w)/r$. Because a disk with radius r and placement error w can be treated as an exactly placed disk with radius $r - w$ [13], we can compensate for the inexact placement by scaling the lattice.

Since in the rigid 6-grid, the grid granularity is $r\sqrt{3}$, the granularity d_6 of the 6-grid accommodating w-bounded placement error is given by: $d_6 = (r - w)\sqrt{3}$. Conversely, if all sensors are free to move in a 6-grid with granularity d_6 and coverage radius r, solving

for w indicates how far the sensors may be allowed to stray from their original positions without violating the coverage guarantee.

For an m-grid, or indeed for any sensor configuration in general, the number of sensors used when we allow wiggle room w is in general scaled by a factor $r^2/(r-w)^2$.

Since given r and w the number of sensors required is affected by the same factor in all cases, and since the rigid 6-grid is optimal, the best solution in the case of inexact placement with bounded placement error is the 6-grid discussed above. From Eq. 9.4.1, the density of sensors $\rho_6(w)$ in a 6-grid which permits a placement error of w is given by:

$$\rho_6(w) = \frac{2\pi r^2}{3(r-w)^2\sqrt{3}} \approx \frac{1.209r^2}{(r-w)^2} \tag{9.4.5}$$

Recalling Eq. 9.4.2, where A is the area of the entire region of interest, the number of sensors $n_6(w)$ needed in this 6-grid is found to be:

$$n_6(w) = \frac{2A}{3(r-w)^2\sqrt{3}} \approx \frac{0.385A}{(r-w)^2} \tag{9.4.6}$$

Similarly, if it were an additional requirement to use a 4-grid, recalling Eq. 9.4.3, the density of sensors $\rho_4(w)$ in such an arrangement allowing a placement error of w is given by:

$$\rho_4(w) = \frac{\pi r^2}{2(r-w)^2} \approx \frac{1.57r^2}{(r-w)^2}. \tag{9.4.7}$$

Recalling Eq. 9.4.4, the number of sensors $n_4(w)$ required in this 4-grid is found to be:

$$n_4(w) = \frac{A}{2(r-w)^2} = \frac{0.5A}{(r-w)^2}. \tag{9.4.8}$$

We note that the ratio of required densities for coverage by 4-grids or 6-grids is invariant with w, i.e., is the same as the efficiency ratio for the two underlying (exactly placed) lattices.

9.5 Robustness to Sensor Failure

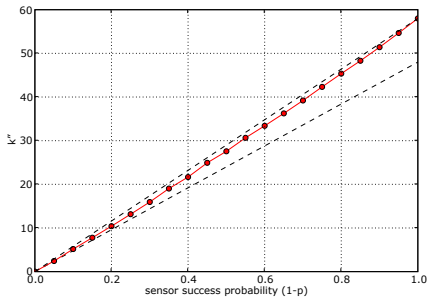
A third benefit of shrinking is robustness to sensor failure. Suppose each deployed sensor fails with probability p. Clearly this will tend to break the full coverage of the hexagon lattice, but shrinking will decrease this tendency. Computing the probability of full coverage in this case is problematic since the probability of full coverage will depend heavily on the size of the relative coverage area. One alternative measure is the probability that a single *local triangle* (i.e., a triangle formed by three mutually adjacent sensors) is covered (or k''-covered for each $k'' \in [1, k']$). Restricting our attention to the disks that fully cover the triangle (in the case of $k = 49$, there are 48 such disks), it is clear that each coverage probability will be lower-bounded by a Bernoulli distribution (shown with a faint dashed line):

Proposition 9.5.1. *The expected coverage multiplicity k'' for a local triangle T is at least $(1 - p) \cdot n_f$, where n_f is the number of sensors fully covering T.*

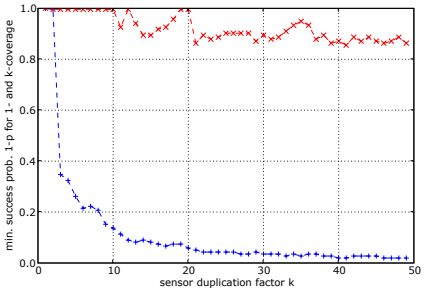
We performed simulations to estimate the coverage obtained when sensors fail (independently) with a fixed probability, as well as the most generous failure probabilities that will yield coverage. Fig. 9.10(a) illustrates the expected multiplicity k'' of coverage for a deployment with $k = 49$ for varying failure probabilities p. Two faint bounding lines are shown, indicating that k'' is bounded below by the stated Bernoulli distribution mean but is strictly convex. Conversely, Fig. 9.10(b) shows the minimum success probabilities required, when the lattice has been shrunk by a factor k, in order to maintain 1-coverage or k-coverage (in expectation).

9.6 Unifying the Three Benefits

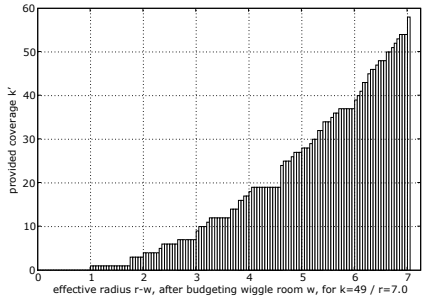
We have examined three separate motivations for shrinking the hexagon lattice (or increasing the sensor radii): obtaining flexibility in sensor placement, increasing the sensor efficiency in obtaining k-coverage, and providing fault-tolerance. Increasing the radius by value w allows for amount w of placement error as discussed in Sect. 9.4. For values $k \in Q'$, this is equivalent to the k-covering where the distance between sensors is shrunk by a factor of $1/\sqrt{k}$. The third benefit to shrinking is to maintain coverage when sensors fail.



(a) Expected coverage multiplicity (k'') with $k = 49$).



(b) Success probabilities for 1- and k-coverage.



(c) Efficiency gain after allocating wiggle room budget.

Figure 9.10: Computational results.

Thus increasing the radius r from 1 to a larger value can be construed as providing

a "redundancy" budget which may be spent in multiple ways. If the enlarged radius r is "spent" entirely on wiggle room $w = r - 1$, then we may only rely upon 1-coverage. Conversely, if r is spent entirely on getting $k' \approx 1.209k$ coverage, then no flexibility in sensor placement is provided. Finally, it can be spent entirely on failure probability (as in the lower curve in Fig. 9.10(a)). We can first of all partition the benefit of r as $r = r_w + r_c$, yielding wiggle room $w = r_w$ and at least k-coverage for $k \geq r_c^2$ (which may, as indicated above, be made probabilistic). A single shrinking factor could provide a range of possible tradeoffs between benefits, with the degree of coverage and the amount of allowable placement error varying depending on how the shrunk lattice is interpreted. Fig. 9.10(c) illustrates the amount of efficiency gain received after allocating some amount $w < r$ to wiggle room, starting with the radius r corresponding to $k = 49$. Notice that in varying the effective radius, we revisit some of the values k shown in Table 9.1 above. For example, for $k = 27$ and $w = 0.613$ we have $r = \sqrt{27} - w \geq \sqrt{21}$, which implies that 24-coverage is obtained.

Second, after allocating wiggle room, there are various potential ways to further divide the shrinking benefit between multi-coverage and failure robustness. Fig. 9.10(a) above, for example, can be interpreted as characterizing the possible ways of allocating the shrinking factor 7 (corresponding to $k = 49$) between allowable sensor failure probability and expected multiplicity of coverage. For a particular value k, two natural trade-off choices are those in which the maximum allowable failure probabilities are chosen that yield either 1-coverage or k-coverage in expectation (see Fig. 9.10(b) for an estimation of these curves, based on random trials). In the case of $k = 49$, these two choices can be extracted from the points on the curve in Fig. 9.10(a) with $x = 1$ or $x = 49$.

In conclusion, we have examined three kinds of benefits between which the shrinking

"budget" can be divided. We emphasize that this budget need not be merely metaphorical. As we saw in Sect. 9.4, the size of the coverage region and the lattice density together determine the number of sensors used; their cost can be significant, in terms of equipment costs, placement, and energy. In this paper we have considered three reasons to pay this price.

Chapter 10

Sensing and Wiggle Radii

10.1 Introduction

In sensor networks, a disk is typically used to model the coverage range of a sensor (see e.g. [25]). In the binary coverage model, a sensor's coverage region is a disk centered on the sensor location. There are a variety of coverage problems which solve for optimal locations for sensor placement, or conversely, given a chosen set of sensor locations, solve for optimal (disk-shaped) coverage ranges, according to some objective function (see e.g. [8]). The problem prompting this chapter is that of assigning radii to sensor points to cover a discrete set of client points, while optimizing some objective function, typically minimizing the sum of the radii $\sum_i r_i$. It is ordinarily assumed that sensors can be (or have been) placed exactly in their *intended* positions. Since sensors are physical objects, however, this assumption of exact placement is only an ideal, which may in realistic situations be violated to various degrees. There may be physical obstructions preventing placement in particular locations, in both urban environments and in nature. More fundamentally, the placement of any physical object is necessarily performed with some amount of error. This

fact yields a complementary class of problems in which we seek to maximize the allowance for placement error, or *wiggle room*, among the sensors. Given a set of positioned unit disks, a dual problem is that of maximizing the sum of the *wiggle radii* $\sum_i w_i$, where w_i is the amount of (directionless) placement error granted to sensor i.

Minimizing r_i and maximizing $(1 - r_i)$. Let us indicate a disk with radius X and wiggle radius y as an (X, y) disk. Then an (R, w) disk is functionally equivalent to an $(R - w, 0)$ disk in the sense that in a given sensor configuration, one (R, w) disk can be replaced with an $(R - w, 0)$ disk, and vice versa, without affecting the coverage guarantee. That is, for any point P of distance greater than $R - w$ from the center of the disk O, there exists a point Q whose distance from O is less than w and whose distance from P is greater than R. As a result, point P is not covered if the center of the disk is on Q (see Fig. 10.1). Although the (R, w) disk will in fact cover a greater area than the $(R - w, 0)$ disk, since it cannot be known *where* this extra area is, there is no advantage to the larger, inexact disk.

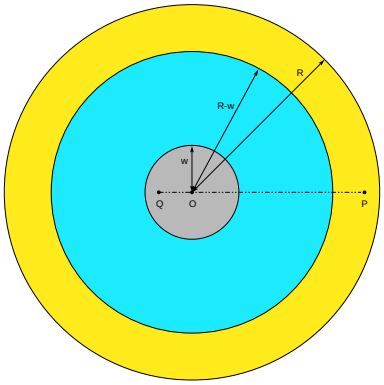


Figure 10.1: Sensing disk and wiggle disk.

When it comes to assigning allowable wiggle radii based on a given set of sensor positions, there are two important differences from the problem of assigning sensor radii. First, the objective function is now slightly different. If the actual radius is normalized

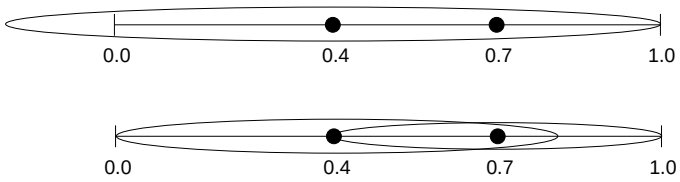


Figure 10.2: *1-d* continuous example

	DISC1D	DISC2D	CTN1D	CTN2D
SUMRAD	nm	in P [46]	FPTAS	FPTAS
SUMWIG	$nm + n^2$	NP-hard/PTAS	-	NP-hard
FAIRRAD	$(nm)^2$	NP-hard	$n \log n$	in P
FAIRWIG	$(nm)^2$	NP-hard	$n \log n$	in P

Table 10.1: Summary of problem settings, many of which interrelate. In some settings, hardness and easiness depend on the strength of general-position or λ-precision [61] assumptions used.

to 1 and we interpret the effective radius then as $r_i = 1 - w_i$, the objective function becomes $\max \sum_i (1 - r_i)$ rather than $\min \sum_i r_i$. Obviously the optimal solutions for the two objective functions will be the same, but the difference in objective functions may affect approximation algorithms significantly. Second, the wiggle radius problem takes for granted that the sensors have fixed (say, unit) radii, some of which will be sacrificed for flexibility. In terms of optimal solutions, the wiggle radii problem is equivalent to the min-radius problem in which there is an upper bound on allowable sensor radii. A *1-d* continuous problem instance comprising two sensors at locations .4 and .7 within a coverage range [0,1], is shown in Fig. 10.2 with two possible solutions. The first solution has radii $< .6, 0 >$ and wiggle radii $< .4, 1 >$[1]; the second has radii $< .4, .3 >$ and wiggle radii $< .6, .7 >$.

Fairness If the goal is only to minimize the sum of disk radii, as a proxy for total sensor energy or cost, an optimal solution might assign a small radius to one sensor and a large

[1]Note that a wiggle radius of 1 is the extreme case in which the sensor can be placed anywhere.

radius to another. Equivalently, one disk could get wiggle radius 1 and another 0, which is clearly unacceptable given the motivation of wiggle room. (Similar motivations would apply if the radii in the sum of radii (SUMRAD) problems were interpreted as individual costs that must be contributed to an agreed-upon joint project.)

One way to partially alleviate this problem would be to assume larger disks. We assume throughout the chapter that *with zero wiggle room assigned, the unit disks will together form a feasible problem solution*. We might make a stronger assumption that even if all disks are assigned wiggle room of, say, .2, full coverage would still be provided. We could then seek to maximize the allotment of the remaining wiggle room, taking $.8$ as the radius bound. (A similar assumption is made for continuous summation (CTNSUM) in Section 10.3.3.)

We also consider fairness as a direct goal. A natural objective in this vein is maximizing the minimum wiggle radius. A generalization of this idea from networking is *max-min fairness* (MMF) [21]. For a maximization problem, an assignment of values is MMF if raising any value x_i would necessarily decrease another value x_j such that $x_j < x_i$, or equivalently, if the vector of assigned values (x_i), ordered by increasing value, is lexicographically maximal. (In Fig. 10.2, the latter solution is MMF.)

Minimizing the sum of disk radii raised to a superlinear power, $\sum r_i^\alpha$ for $\alpha > 1$, rather than minimizing the simple sum of radii, goes some way towards encouraging fairness. A larger α will more strongly encourage fairness, since e.g. this make one large radius more expensive than two small ones, but it is easy to construct examples for any fixed α that result in unfair assignment. The minimization problem was first shown to be *NP*-hard for $\alpha \geq 2$ [22], and later for $\alpha > 1$ [8]. The discrete MMF problem, which can be seen here as the limiting case of this problem as $\alpha \to \infty$, i.e., the ℓ_∞ norm, can be shown to be *NP*-hard by modification of the hardness argument in [8].

Explanation of Table 1 and Problem Names. Combining the choices of *1-d* v. *2-d*, wiggle maximization v. radius minimization, summation v. fairness, and discrete v. continuous we obtain 16 problem settings (see Table 10.1). We refer to a given problem setting by an abbreviation such as DISC1DSUMRAD or CTN2DFAIRWIG. Many of the problem solutions are related or identical. An abbreviation such as 2DFAIR omitting some of the four properties is understood to indicate all compatible problem subcases, ranging over all choices for the omitted attributes.

All present entries in Table 1, apart from DISC2DSUMRAD indicate our results. Note that there are several holes in the table. For some settings, we know only hardness or approximation schemes but not both. We give no results for CTN1DSUMWIG.

Summary of Results and Techniques. We relate the placement problem for unit disks with fixed uniform wiggle radii for guaranteed coverage to coverage solutions for boolean disks. Given a set of disk sensors, we show for the *2-d* setting how to obtain FPTASs for the discrete wiggle radius problem and the continuous sensor radius problem. We also show that the discrete wiggle radius problem is *NP*-hard. In the *1-d* setting, we observe that the wiggle radius problem can be solved optimally by known algorithms and we give a faster dynamic programming algorithm for the (linear) sensor radius problem. Finally, we consider max-min fairness in the assignment of sensor or wiggle radii in *1-d* and *2-d*. Although the discrete problem is *NP*-hard, we give an exact polynomial-time algorithm for the continuous problem, assuming the sensors are in general position.

In some cases, the inexact placement problems reduce easily to known problems, while in others the nature of the problem changes significantly. We study these problems in both *1-d* (on the line) and *2-d* (in the plane). In the *continuous* setting, the object is to cover a continuous, convex region; in the *discrete* setting, there is a given finite set of points

(clients) to cover. The problem instance consists of the locations of n sensors and (in the discrete case) m clients.

Several algorithms for the discrete setting (both *1-d* and *2-d*) involve "pinned" disks. A pinned disk is centered on sensor i, with r so that the disk just touches some client j, i.e., we have $r_i = d(i, j)$, where $d(i, j)$ is the separating distance. It is well known that any optimal solution to the min-radius problem will consist of pinned disks. The same applies to the max-wiggle problem, though not all possible pinned disks will exist.

10.2 Related Work

There have been efforts in various communities, e.g. in computational geometry, approximation algorithms, sensor networks, to characterize, find approximate solutions to, and analyze the intrinsic properties of, coverage problems involved binary sensors. We briefly refer to some of this work.

Covering a large convex region (ignore edges, or assume the region of interest is the convex hull of the sensor locations) with a minimal number of disks of radius R, is solved optimally by a hexagonal lattice arrangement of disks (see Figure 1.a [71, 104]).

[66] discusses the more general concept of a sensor's *wiggle region*, i.e., the point set of locations for the disk center that are consistent with the global coverage guarantee (see the shaded areas of Figure 1.b), assuming other sensors are constrained by their own wiggle regions, without leaving any point uncovered. In lattice arrangements, wiggle regions will be approximately disk-shaped. Moreover, the motivation for wiggle room is bounded placement error, which is likely to be non-directional or *isotropic*. We therefore limit ourselves in this chapter to wiggle *disks*.

For the problem of positioning binary unit-disk sensors to cover a large continuous

field, it is well known that a hexagon grid configuration is optimal. Recently there has been some work [55] on positioning sensors whose coverage is probabilistic, based on distance. A related but significantly different problem is to decide which, among a set of positioned sensors, to turn on to cover the region. Hochbaum & Maass [57] gave a well known PTAS for several disk packing and covering problems in the plane, based on "grid-shifting". This technique was extended to Maximum Independent Set and Vertex Cover on non-unit disk graphs by Erlebach et al. [37], in a recursive dynamic programming procedure that uses a multiple grids varying in granularity, corresponding to disks of different sizes.

The Erlebach et al. technique was used by [74] and [22] for the discrete sensor radius minimization problem, in which the goal is to cover a discrete set of clients in the plane. Optimal dynamic programming algorithms for the *1-d* setting have been given by [74, 8, 73]. [74] gives a dynamic programming (DP) algorithm to solve the problem (optimally for the generalization of $\alpha \geq 1$) optimally in $O((n+m)^3)$ time, where m = #clients and n = #sensors. Lenchner [73] recently gave a faster $O(m^2 + nm)$ DP. Several other related problems have been considered; see [8] and references therein.

The *NP*-hard Facility Location problem [95] is similar, with the difference that one pays for the coverage of each client, based on the distance to the covering server, rather than only once for the sensor radius. More closely related are clustering problems such as k-Median, in which the radii of k clusters, together containing all points, are minimized. It was recently shown [46] that the *2-d* discrete sensor radius assignment problem (and related clustering problems) can be solved in polynomial time.

10.3 Optimizing the Sums of Wiggle and Sensor Radii (SUM)

10.3.1 Optimizing Radii Sums in Discrete *1-d* (DISC1DSUM)

In *1-d*, sensors and clients lie on the line, with m = #clients and n = #sensors. The Lenchner algorithm [73], which runs in $O(m^2 + nm)$, can be modified to observe the radius bound, by removing larger disks from the set of pinned disks considered, so it can be applied to DISC1DSUMWIG as well. For DISC1DSUMRAD, we next give an optimal DP algorithm that runs in $O(nm)$ time. We first define some notation: set S contains the indices of the sensor nodes and set C contains those of the client nodes; v_i is the ith node (either sensor or client); $d_{i,j}$ is the distance between nodes i and j; $C_{j,i}$ is the pinned circle centered on sensor j with radius $d_{i,j}$; $r(i)$ is the radius assigned to node i, which is necessarily 0 if v_i is a client; for a client i and a sensor j with $j < i$, $f(i, j)$ is one less than the index of the leftmost client contained in circle $C_{i,j}$.

The DP forward procedure (see Figure 2) fills two arrays, c and $\hat{c}$, of solution costs. (For simplicity, we explain only how to compute the optimal cost, omitting the computation of the optimal solution.) $c(i)$ is the optimal solution cost for the problem instance consisting of the first i nodes; $\hat{c}(i)$ is the optimal cost for the instance consisting of the first $i-1$ nodes and *the ith node replaced with a sensor* (which we call a *pseudo-sensor*). ($\hat{c}(i)$ is redundant whenever node i is itself a sensor, but we include it here for notational convenience.) The DP is based on this observation: if the only remaining clients to cover lie to the left of a circle with radius r_1 centered on v_1, then increasing v_1's radius from r_1 to $r_1 + r_2$ is equivalent, in both cost and coverage benefits, to adding a new sensor with radius r_2, positioned at a distance r_1 to the left of v_1. Intuitively, the DP works as follows. Given a new sensor v_i, either it is ignored or it is given radius $d_{i,i-1}$ and node v_{i-1} is replaced with a sensor

Algorithm 7 DP Algorithm

for each pinned circle $C_{j,i}$ **do**
  $L(j,i) \leftarrow$ the index of the leftmost client contained in $C_{j,i}$
  $f(j,i) \leftarrow L(j,i) - 1$
end for
$c(0) \leftarrow 0$
$\hat{c}(0) \leftarrow 0$
for $i = 1$ to $m + n$ **do**
  $\hat{c}(i) = \min(c(i-1), \hat{c}(i-1) + d_{i,i-1})$
  if $v_i \in S$ **then**
    $c(i) = \hat{c}(i)$
  else
    $c(i) = \min_{j<i, v_j \in S} \{d_{j,i} + c(f(j,i)), d_{j,f(j,i)} + \hat{c}(f(j,i))\}$
  end if
end for

if necessary. Given a new client, it will be covered by some prior sensor v_j. Either that sensor will be given radius $d_{j,i}$, or it will be given a radius just large enough to reach the previous node (more than distance $d_{j,i}$ to the left of v_j), which is replaced with a sensor if it is not one. Of course, such pseudo-sensors are not really used. During the backtracking phase, each non-zero radius assigned to a replacement sensor will be *shifted to* the following sensor. The backtracking process can then recover the actual optimal solution in linear time.

We now examine the handling of a new client in greater detail. For each such client i and prior sensor j that we consider for it, two options are considered in the recursion for the circle centered on j: either it just touches (is pinned by) i, or it is somewhat larger:

Option 1: When sensor j's circle C just touches client i, let $k = L(j,i) - 1$. If node k is a client, then it lies prior to C, by definition of $L(j,i)$. If it is a sensor, it may actually lie within C, but this is fine. What is important is that we combine C with a solution for all the nodes prior to C (including extra sensors does no harm). In this case the solution cost

is $d_{i,k} + c(j)$.

Option 2: When sensor j gets the larger circle C', let k be the index of the rightmost node lying distance $> d_{ij}$ to the left of sensor j (i.e., missed by the circle C). There is no benefit to expanding C to touch a previous sensor, and anyway it could actually happen that the node prior to $L(j, i)$ is a sensor *inside* C, but there is no harm in evaluating this choice. The important strategy we want to try is expanding the circle to reach the most recent node, since if it is a client, covering it with the new circle may yield a cheaper solution. In this case the solution cost is $d_{k,j} + c'(j)$.

Lemma 10.3.1. *The DP forward procedure runs in $O(nm)$ time.*

Proof: It is known that the leftmost clients of all nm pinned circles can be found in time $O(nm)$ [73]. To obtain the bound, we observe that in handling a new node, two subcases are considered if the new node is a sensor, and $2m$ subcases are considered if it is a client. From the discussion above, considering each such case can be done in $O(1)$. □

We now prove correctness.

Theorem 10.3.2. *The DP forward procedure computes the optimal solution value.*

Proof: (sketch) By induction. In the base case of zero nodes, the optimal solution (zero radius) is trivially obtained. Now assume the optimal solution is obtained for all problem instances of length at most k. Then consider one of length $k + 1$. Suppose v_{k+1} is a sensor. In the optimal solution, $r(k+1)$ must be either 0 or $\geq d_{k,k+1}$. If the latter case, the solution cost would be the same if an inverse-shift were performed, i.e., $r(k) = r(k + 1) - d_{k,k+1}$, $r(k + 1) = d_{k,k+1}$. $\hat{c}(k)$'s optimality then implies $c(k + 1)$'s. On the other hand, suppose v_{k+1} is a client. In an optimal solution, v_{k+1} is covered by some sensor v_j, with radius either $d_{j,k+1}$ or larger. In the latter case, the solution cost would again be unchanged by an inverse-shift, and so optimality again follows. □

Unfortunately, the DP algorithm above does not apply to DISC1DSUMWIG, because of the bound on allowable sensor radii. The algorithm could be modified so that, in handling the case of a new client, the previous client is only considered if it is near enough. The difficulty lies in the handling of the pseudo-sensors. The handling of a new client reduces to an indeterminate number of pairs of precomputed subcases, half of which involve a pseudo-sensor. When the subcase ending at such a pseudo-sensor is solved, there is no way to know which latter sensor(s) it will be treated as an extension of, and therefore no way to know what bound on its radius to use.

10.3.2 Optimizing Sums in Discrete *2-d* (DISC2DSUM)

Since DISC2DSUMRAD is known to be solvable, we turn to the wiggle radii problem, of maximizing $\sum_i 1 - r_i$, subject to $r_i \in [0, 1]$. The *superlinear* version of the corresponding sensor radii problem is known to be *NP*-hard; a hardness result for the (linear) wiggle radii problem is implicit in the superlinear hardness proof of Alt et al. [8].

Proposition 10.3.3. DISC2DSUMWIG *and* bounded DISC2DSUMRAD *are* NP*-hard.*

Proof: Alt et al. [8] reduce from PLANAR 3SAT to DISC2DSUMRAD for $\alpha > 1$, by constructing a sensor/client graph with an alternating chain of disks for each 3SAT variable, and a gadget for each 3SAT clause. There are two important steps in the proof. First, the chains are drawn finely enough (based on the value α) that the optimal solution will choose every other disk in each chain ("even" disks or "odd" disks), rather than larger disks subsuming more than one of the "chain" disks. Second, for the same reason, one of three smaller circles will be chosen in each gadget, to cover a client at its center, rather than enlarging one of three chain disks meeting at the gadget. We can obtain these two properties by placing the disk upper bound to be precisely the radius of the chain disks. □

A PTAS for maximizing wiggle room does not *immediately* follow from the existence of a PTAS for sensor radius minimization. A natural strategy is to run the sensor PTAS to obtain a set of sensor radii and then to assign the wiggle radii $w_i = 1 - r_i$. But nothing prevents the PTAS from assigning a radius greater than one, which absurdly implies a negative wiggle radius. Moreover, an approximation guarantee for one problem does not guarantee an approximation bound for the other. Suppose e.g. $\epsilon = 1/2$, $OPT_{sens} = 1/2$, and $ALG_{sens} = 1$. Then $OPT_{wig} = 1/2$ and $ALG_{wig} = 0$. Nonetheless, by adapting the sensor PTAS, a wiggle PTAS can be obtained.

The polynomial running time of the Lev-Tov & Peleg DP is obtained by bounding the number of "semi-disjoint" disks (neither containing the other's center) of roughly equivalent size will fit in or overlap a grid cell. Intuitively, the algorithm can limit itself to semi-disjoint disks because any solution with two non-semi-disjoint disks could be improved by increasing the size of one disk and removing the other. Of course, with a bound on allowable disk size, this is not always possible. If we reasonably assume that the graph is a λ-precision graph [61], for some fixed $\lambda > 0$, then a PTAS is obtainable.

Proposition 10.3.4. *For graphs of sensors and clients drawn at λ-precision for some fixed $\lambda > 0$, there exists a PTAS for* DISC2DSUMWIG.

Proof: Due to λ-precision, the number of nodes within a fixed-size cell will be bounded by a constant, and so the grid-shifting PTAS technique directly applies. □

10.3.3 Optimizing Sums in Continuous *1-d* and *2-d* (CTNSUM)

In this section, we consider the problem setting in which sensors are discrete and must be assigned radii (without a bound) to cover a continuous region. Ignoring edges, this region

is assumed to be the convex hull of the sensor points in *2-d*. In *1-d*, this is the interval from the leftmost sensor to the rightmost.

In *1-d* there exist very simple approximation algorithms. Let the length L of the interval to be covered be normalized to $L = 1$. Then the optimal solution cost must lie between $.5$ and 1. Therefore assigning radius 1 to the leftmost sensor immediately yields a constant-time 2-approximation. One ideal situation is that there exists a sensor exactly in the middle of the interval. Assuming sensor locations are given as a sorted list, this suggests an $O(\log n)$-time algorithm (*Centermost*): find the sensor nearest to the interval center, and assign it a radius large enough to cover both ends.

Proposition 10.3.5. Centermost *is a 1.5-approximation algorithm.*

Proof: If there is a sensor in the range [.25,.75], then ALG pays at most $.75$ while OPT pays at least $.5$. If not, then ALG pays at most 1 while OPT pays at least $.75$. □

For discrete sensors, optimal solutions can be found for *1-d* and near-optimal solutions can be found for *2-d*. By reducing to the discrete setting, near-optimal solutions can be efficiently found for both *1-d* and *2-d* continuous settings.

Proposition 10.3.6. *For λ-precision instances,* CTN1DMINRAD *is solvable in polynomial time.*

Proof: (sketch) We first claim that in any problem instance, there exists an optimal solution with no overlap between any two sensors' assigned disks. Given an arbitrary solution, overlaps can be eliminated one by one, by shrinking the second disk and then increasing the following disk, without changing the total cost of the solution.

Now, suppose all sensors lie at locations specified at λ-bit precision (for constant λ), or equivalently at integral values in a scaled range. Since there is an optimal solution having

no overlap, in such a solution all sensor radii are λ-bit values. Now we place discrete sensors at each multiple of $2^{-\lambda-1}$. Known algorithms for DISC1DSUMRAD, slightly modified so that only b-precision disks, as opposed to $\lambda+1$-precision, but including all clients, will now solve this problem optimally. □

Proposition 10.3.7. *There exists an FPTAS for* CTN1DSUMRAD.

Proof: We want to find a solution with cost at most $(1+\epsilon)\cdot OPT$. We know the optimal solution will be at least $1/2$. Therefore we place equally spaced discrete clients along the interval so that the separating distance is $\epsilon_2 = \min(\epsilon/(2n), \lambda)$, and we solve this discrete problem optimally in poly time. We now increase all sensor radii by amount ϵ_2, which will fill in any coverage gaps, at a cost of at most $\epsilon \cdot 1/2 \leq \epsilon \cdot OPT$. □

Note that the dependency on ϵ is quite reasonable. In the case of $\epsilon = 1/n$, for example, the total running time is only $O(n^3)$.

Proposition 10.3.8. *There exists an FPTAS for* CTN2DSUMRAD.

Proof: We construct an instance of the discrete problem, which is in P. The continuous problem instance consists of a set of sensor locations and a convex coverage region, which we assume is specified by a set of points. Given this, we produce an instance of the discrete problem. It has the same sensors as does the continuous instance; its client locations form a mesh (square, say) over the coverage region.

First, let $D = \max_{s_1 \neq s_2 \in S} d(s_1, s_2)$, which can be thought of as the diameter of the region. Notice that the optimal solution value will be at least $D/2$ and at most $4D$. Let ϵ be the (multiplicative) error parameter to the continuous PTAS.

Let d be the separating distance of the mesh, i.e., the distance between a client and its neighbors. Then we draw a mesh of clients fine enough so that $d < \min(D\epsilon/m, \lambda)$, where

m is the number of sensors. Then the number of clients is $\Theta((D/d)^2) = \Theta((m/\epsilon)^2)$, which is polynomial in m. We now run the optimal algorithm on the discrete problem instance. The resulting solution covers all clients, but it may leave uncovered some small region between them. Since any such region is small enough to fit within a $d \times d$ square, we modify the resulting solution by increasing the radii of all sensors by amount d. The resulting solution will cover the entire region.

Then since $md \leq D\epsilon \leq \epsilon \cdot OPT$, we have that

$$ALG \leq OPT_{dis} + md \leq (1+\epsilon)OPT$$

Therefore the algorithm achieves the required approximation guarantee.

The subroutine runs in time bounded by some polynomial $p(nm)$. The running time of the continuous FPTAS will be bounded by $p(m(m/\epsilon)^2)$, which indeed is polynomial in m and $1/\epsilon$. □

Because the objective is minimization, this reduction does not apply to CTNSUMWIG.

10.4 Fairness in Wiggle and Sensor Radii (FAIR)

The second difference between SUMRAD and SUMWIG is that the latter implicitly has a bound on the sensor ranges, meaning that optimal solutions for the two problems could diverge, since e.g. an optimal solution for the former could consist of a single huge disk. This difference is irrelevant to FAIR, since first of all the largest assigned radius should be as small as possible. Therefore for each sub-setting FAIRRAD and FAIRWIG are really the same problem, and so we do not distinguish between such pairs.

10.4.1 Fairness with Discrete Clients (DISCFAIR)

One of the simplest algorithms for DISC1DSUMRAD is an $O(nm^2)$ algorithm that, for each increasing prefix of j client, finds the cheapest solution to cover them by considering each of the $O(nm)$ pinned disks covering the ith client, combined with the optimal solution (already computed) for the clients lying to the left of that disk. This algorithm can be adapted to the metric of fairness at the cost of one order of magnitude, since comparing two vectors of radii can be done in m time. Thus we have:

Proposition 10.4.1. DISC1DFAIR *is in P.*

Proposition 10.4.2. *Without restricting assumptions,* DISC2DFAIR *is* NP*-hard.*

Proof: We obtain the result by examining the hardness result of Alt et al. [8]. In that reduction, a sensor/client graph is constructed in which a minimum-cost solution will choose half the "chain" disks of each chain and at least one of the three small disks of each gadget. Such a solution will in fact be max-min fair, since removing or shrinking any of the chosen chain disks or small gadgets would necessitate the inclusion of a still larger disk. □

With a strong enough general-position assumption, e.g. if random noise is applied to all sensor and client positions, the sort of MMF techniques used for CTNFAIR can also be made to apply to DISCFAIR. We omit the details in this extended abstract.

10.4.2 Fairness with a Continuous Region (CTNFAIR)

In the continuous setting, however, a MMF set of radii for n sensors can be found in polynomial time, both in *1-d* and in *2-d* (in the latter case if the sensors are in general position, i.e. no set of more than three points lie on a circle; or that the resulting Voronoi diagram

Algorithm 8 MMF Algorithm

```
1: insert into the heap the initial coverage cost r(p) of each subregion
2: while the heap is not empty do
3:     perform a delete-min, removing subregion p
4:     assign r(p) to one or more of p's sensors
5:     update the coverage costs of other subregions as needed
6: end while
```

is nondegenerate). The algorithm for the two settings is based on the same schema. As above, we assume that the configuration of sensors is feasible in the sense that given zero wiggle room, full coverage will be achieved. Given this, the optimal max-min fair sensor assignment and min-max fair wiggle assignment solutions will be identical. Therefore we do not distinguish between the two problems in the following. The algorithm schema is shown above.

Let the region of interest be partitioned in some way into subregions, each of which is defined by a set of bordering sensors. For each not-fully-covered subregion, let its *current cost* be the maximum radius value required to cover a currently uncovered point within it, given the previously assigned radii. Let the *critical region* be the hardest such remaining uncovered point within the entire region. Then we must show three things: that the critical region can be found in poly time; that the radii to cover it can be chosen in poly time; and that if there are ties for critical region then the ordering of tie-breaking does not matter. In this case, the resulting radii assignment will be fair.

Theorem 10.4.3. *If the ordering of tie-breaking does not matter, then MMF is achieved.*

Proof: We prove by induction, over the length of prefixes of the subregion list. For the base case, it is clear that the first step will minimize the maximum radius. Let R_i be the subregion (some of) whose sensor(s) are assigned radii in round i. For R_i, reached after assigning radii (to n_i sensors) that form a prefix of a MMF vector, we must show that the

radii assigned in round i also satisfy MMF. If there are no ties, or if the ordering of tie-breaking does not matter, then this immediately follows. Since we consider all remaining subregions, the new values added to the MMF vector are minimal, and the larger vector is again MMF. □

Fairness in *1-d* (CTN1DFAIR) In *1-d*, the interval is partitioned by the sensors into subintervals, each of which is defined by two sensors. The cost $c(p)$ of each subinterval p is set to half its width. These values are placed into the heap in $O(n \log n)$. Clearly the first subinterval $\hat{p}$ extracted from the heap is the hardest to cover. In an MMF radius assignment, both of p's sensors must be given radius $\hat{p}$; if either was less the other would have to be greater. Of course, it is possible for either of these radius assignments to cover other subintervals partly or entirely. Consider p's right sensor. It will completely cover 0 or more subintervals and then partly cover 0 or 1 additional subinterval. Searching the array of sensor locations, the last subinterval it (partly) covers can be found in time $O(\log n)$. All fully covered subintervals (with cost 0) can be removed from the heap. If the (possible) partly covered subinterval is more than half covered, then the difficulty of it in the heap will be reduced to the size of the portion left uncovered (and when chosen, only its right sensor will be assigned); otherwise, its difficulty will be unchanged.

Clearly assigning radius $\hat{p}$ to *all* sensors will minimize the maximum. Instead, we repeatedly search for the uncovered interval p requiring the next largest radius. Since the difficulty of the remaining subintervals is updated after each selection, the selection itself is simply a delete-max operation on the heap, for a total of $O(n \log n)$.

If there are no ties, then each move is forced, and it is clear that the result is MMF. Suppose that at some point there are two subintervals p and q that tie as most difficult, i.e., with $r = r(p) = r(q)$. Then we claim the order in which they are chosen does not matter.

Suppose interval p lies before q, separated by some distance $d \geq 0$, and suppose p is chosen first. If $d = 0$, there are three sensors involved, say 1,2,3. (Assume wlog that none has yet been assigned a radius.) When p is chosen, 1 and 2 are assigned r; when q is later chosen, 3 is assigned r. If $d > 0$, there are four sensors involved, say 1,2,3,4. When p is chosen, 1 and 2 are assigned r; when q is later chosen, 3 and 4 are both assigned r because sensor 2's coverage will not reach the center of interval q.

Fairness in *2-d* (CTN2DFAIR) We follow the same algorithm schema in *2-d*, but implemented differently. In place of subintervals, we work with Delaunay triangles. Whereas in *1-d* newly assigned radii meet at a subinterval's center, in *2-d* they meet at the Voronoi point at the triangle's center. For each Voronoi vertex p (equidistant between *three* sensors), its cost $c(p)$ is the length of its Voronoi edges. Among all Voronoi vertices, take the point $\hat{p}$ of maximum cost $c(\hat{p})$. Then, because decreasing any any of $\hat{p}$'s three radii would force another to increase, we have the following:

Lemma 10.4.4. *In a MMF radius assignment covering a neighborhood around $\hat{p}$, all three neighboring sensors must be given radius $c(\hat{p})$.*

Once again, assigning value $c(\hat{p})$ to *all* sensors suffices to minimize the maximum radius. For a not-fully-covered Delaunay triangle, its *current cost* is the maximum radius value required to cover a currently uncovered point within it, given the previously assigned radii. We repeatedly search for the next critical region. Therefore it suffices, in each round, to determine each Delaunay triangle's cost, which is much more complicated than in *1-d*. There are several issues to consider. First, one or two of the triangle's sensors may already have been assigned a radius. Second, part or all of the triangle may already be covered by sensors external to it. Third, there could even be more than one continuous region of the triangle uncovered. We consider each case in turn.

Lemma 10.4.5. *If no external disks intersect with the triangle, its current cost can be found in poly time.*

Proof: If none of q's three sensors has been assigned a radius, it is handled the same way as the first triangle, by assigning $c(q)$ to the three. If at least one radius has been assigned, then the other(s) are given appropriately smaller radii so that q's Delaunay triangle is covered. If exactly one of the three sensors (say, A) has already been assigned a radius, then (identical) the radii values given to the other two (say, sensors B and C) will be chosen so that the three sensors' disks meet at one point, which will lie upon the Voronoi edge separating the regions of B and C. (This point can be found by elementary geometry.) If two of the three have been assigned (say, A and B), the third is chosen so that it intersects with the closer of the two intersection points of A and B. □

Lemma 10.4.6. *If there is only one uncovered region in the triangle, its current cost can be found in poly time.*

Proof: Now assume the triangle is (partly) covered by *other* sensors, i.e., not those located at the triangle's vertices. The $O(n^2 \log n)$ Huang & Tseng algorithm [58] for computing the number of times that a region is covered by non-unit disks can be used to find coverage holes in a region and the border of the covered area, which is composed of a series of (at most n) arcs. What interests us here is when the border of coverage intersects with the triangle. If the entire triangle is covered, then it is disregarded. In the second case, a series-of-arcs border will pass through. There are now several subcases. If the triangle's Voronoi point q (its center) is not covered, then we proceed as above, ignoring the partial coverage. If it is covered, it may be covered *from one direction* or *from two*. Let the triangle's three vertices be A, B, C. Let d_{Aq} be the distance between points A and q, and let

C_{Aq} be the disk of radius d_{Aq} centered at A. Say that q is covered *from direction* A if the intersection of the triangle and C_{Aq} is contained in the cover.

First, suppose that q is covered from only one direction, say A. Then we extend the line segment $A-q$ until it intersects with the border at some point q' (by elementary geometry). We then give sensors B and C the (identical) radii that allow them to intersect at q', which will completely cover the triangle. Clearly the meeting point of B and C cannot lie outside the coverage region, so if the radii are minimized it must lie on the border. Also, if it lay anywhere else on the border, one radius would decrease and one would increase, violating MMF. Second, suppose that q is covered from two directions, say A and B. Then C is given the radius that makes it intersect with the farthest away border crossing point contained in the triangle, thus covering the triangle. □

Lemma 10.4.7. *Even if T has > 1 uncovered regions, its current cost can be found in poly time.*

Proof: In this case, it could indeed happen that different minimal radii are required to cover the entire triangle (not considering others). In such a case, we only treat the largest of these radii as being the required amount to cover this triangle. If this triangle is chosen as the critical one, only those radii are assigned, and the triangle (now more fully covered) waits for a future round to be fully covered. □

We note the following two properties of the procedure: in each round, the assigned (one, two or three) radii are identical; no previously assigned sensor radius will ever grow larger. Also, for reasons similar to those in the *1-d* case, the order of tie-breaking will not matter. We conclude by showing running time.

Theorem 10.4.8. *The MMF algorithm runs in polynomial time.*

Proof: The Voronoi diagram computation (using Fortune's algorithm e.g.) is performed in $O(n \log n)$ time. In each round, the costs of each triangle can be found in poly time. Even if these computations, as well as the computation of the union of disks, are done from scratch, each round takes poly time. Since in each round at least one sensor is assigned its (final) radius, the total running time is therefore polynomial. □

10.5 Conclusion

In this chapter we considered a large family of interrelated problems. The solutions for some problems involve general-position of precision assumptions, of varying strength. One future research direction is in weakening these assumptions. Perhaps the most interesting open problem is to efficiently approximate DISC2DSUMWIG and *bounded* DISC2DSUMRAD. Bounded DISC2DSUMRAD is arguably much more realistic than the standard version, in which a single huge sensor radius could be used to cover the entire region. Other open problems include finding an optimal algorithm for CTN1DSUMRAD and extending SUMWIG to non-unit disks. For this last setting interesting objective functions might involve the relative amount of wiggle room, i.e. w_i/r_i.

We emphasize that the second change can also reasonably be combined with the original objective function. In the standard sensor radius minimization problem, it is always a feasible solution to make one sensor disk large enough to cover all clients. For large coverage areas, assuming arbitrarily large sensor ranges is unlikely to be realistic. We call this the *bounded* SUMRAD problem.

Other potential directions for future research include the so-called 1.5 dimension (in which sensors xor clients are constrained to one dimension), and probabilistic wiggle room, which can be reduced to probabilistic disks with certain distributions.

Chapter 11

Pan and Scan

11.1 Introduction

Finally, in this chapter, we study a problem combining sensor coverage and assignment, involving cameras. cameras and targets. In the core geometric problem setting, cameras deployed to observe targets in the field may both swivel and adjust their focal width in order to observe one or more targets. That is, for each camera we choose an orientation and a zoom factor. By enlarging a camera's focal width (and thus its angle of view), we allow the camera to observe more target locations, at the cost of correspondingly reducing the quality of the resulting image; conversely, by zooming in on a small area, the camera will record better quality images of the targets therein, while sacrificing targets lying outside this area. (Ideally, these targets will be covered by other cameras.) The objective, informally speaking, is then configure the cameras in order to observe as many targets as possible, with the highest possible precision.

11.1.1 Motivation and Model

More formally, given are n camera locations in the field and m target locations to observe. (See Figure 11.1.1.) For each camera there are two parameters to set: the viewing direction ϕ and the viewing angle ψ, which assignment will allow the camera to observe all targets in the cone defined by angles $\phi - \psi$ and $\phi + \psi$ and the camera position P. The quality of the a camera c's observation of a visible target t depends on both the distance between c and t and on ψ. The specific deterioration function may very by camera hardware, but for a camera producing rectilinear images, the field of view[1] is proportional to the distance between camera and scene, specifically:

$$\frac{o}{d} = \frac{i}{f}$$

Here f is the focal length, i is the (constant) image dimension, and d is the distance. In this case, the object dimension, i.e., the field of view, is $o = di/f$. An equation of this form holds in both horizontal and vertical dimensions. We are interested in the image quality, which then depends on both d and f:

$$u(s,t) = f^2/d(s,t)^2$$

That is, the observation quality varies inversely with the distance squared and directly with focal length squared. Increasing the focal length, however, decreases field of view $o = di/f$. Thus there is a tradeoff between the angle of the viewing cone and the quality. Each such cone is infinite in area, but the imaging quality degrades as the cone stretches out to infinity. Narrowing the cone thus extends the range of imaging at a given quality farther out in the cone.

[1] `http://en.wikipedia.org/wiki/Field_of_view`

With this tradeoff between imaging quality and scope in place, we can now define an optimization problem. Given a set of target locations and a set of placed sensors, the goal is to configure sensors so as to maximize the total sensing quality. How to encode the assignments? Each possible direction/angle assignment for a sensor will cover some *contiguous* subset of the targets. Although there are infinitely many possible angle/focal length choices, only a finite, polynomial number of such choices need be considered. Choosing the focal length and the angle ϕ is equivalent to choosing the angles ϕ, ψ. For each sensor, we may assume that these angles are *tight* in the following sense.

Definition 11.1.1. *A* pinned cone *(or just a* cone *when clear) is an assignment to a camera of angles so that its cone intersects a client on both sides (possibly the same client).*

For each camera, we can restrict our attention to its $O(n^2)$ possible pinned cones. (The value of any non-tight cone can only be improved by narrowing it until it goes tight.) In order to ensure finite utility values for all possible cones, we assume that each target is itself some distance C wide, so no cone's angle will shrink to 0.

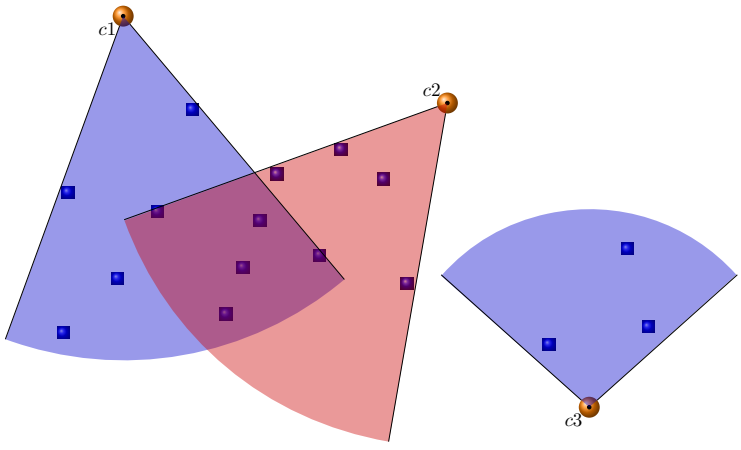


Figure 11.1: Three cameras configured to observe many targets.

ADD aggregation. If a target is covered multiple times, a simple option for tallying its utility is simply to accumulate it in an additive way, as we do in the Integer Program formulation 11.1.1.

$$\begin{aligned} \max \quad & \sum_{i,k \in \sigma(i)} x_{i,k} \cdot \sum_{j=1}^{m} f_k^2 / d(c(k), j)^2 && && (11.1.1) \\ \text{s.t.} \quad & \sum_{k \in \sigma(i)} x_{i,k} \leq 1 && \forall\, i \\ & x_{i,k} \in \{0, 1\} && \forall\, j, k \end{aligned}$$

In this formulation, $\sigma(i)$ indicates the set of indices of the cones based at camera i. The sole constraint set indicates that no camera receives more than one cone. Note that f_k and $d(c(k), j)$ are both constants specifying the problem instance, the focal length of pinned cone k (which is based at camera $c(k)$) and the distance between target j and camera $c(k)$, respectively.

Proposition 11.1.1. *The summed utility formulation given in Formulation 11.1.1 is solvable optimally in polynomial time.*

Proof: For each camera, choose the highest value pinned cone, by enumeration.

MAX aggregation. In some settings, however, accumulating utility from an arbitrary number of cameras is unrealistic. In the extreme case, no aggregation is permitted, and in which case only one camera capturing a target within its cone as viewing the target would be counted when evaluating solution quality. Without loss of generality, it may be assumed that under a given cone configuration, each target is observed only be the camera producing the highest quality observation of it. In other words, in this formulation utility

is aggregated by the MAX function rather than by SUM. This setting is captured by IP formulation 11.1.2.

$$\begin{aligned} \max \ & \sum_{j,k} x_{j,k} f_k^2 / d(c(k), j)^2 && (11.1.2) \\ \text{s.t.} \ & \sum_{k} x_{j,k} \leq 1 && \forall\, j \\ & x_{j,k} \leq y_k && \forall\, j,k \\ & \sum_{k \in \sigma(i)} y_k \leq 1 && \forall\, i \\ & x_{j,k}, y_k \in \{0,1\} && \forall\, j,k \end{aligned}$$

The first constraint set ensures that we do not get credit for covering any target with more than one cone. The second ensures that we do not get we cover targets only with cones that are actually chosen. The third prevents our choosing more than one cone for any camera.

11.1.2 Contributions and Assumptions

We introduce several variants of the camera configuration problem. We show that the ADD variant and a geometrically constrained version of the MAX variant are both optimally solvable in polynomial time (Section 11.3). We prove that the unconstrained MAX variant is NP-hard (Section 11.4). We present two 2-approximation algorithms, one centralized and one distributed in an asynchronous model, and also note a setting that admits a PTAS. For a synchronized distributed setting, we give a 2-approximation protocol and a $(2\beta)/(1-\alpha)$-approximation protocol (for all $0 \leq \alpha \leq 1$ and $\beta \geq 1$) with the stability feature that no target's camera assignment changes more than $\log_\beta(m/\alpha)$ times (Section 11.6). We also

discuss the running times of the algorithms and study the speed-ups that are possible in certain situations.

Although our motivating example problems above have their objective functions defined specifically in terms of distance squared and focal length squared, our algorithms presented below have more general guarantees. To obtain optimal algorithm correctness and constant approximation guarantees, we only assume that the quality of a camera's measurement of a target declines monotonically as either the distance or the field of view grows. Only in the case of the running time speed-ups do we assume that the measurement value is the product of separable monotonic functions of distance and field of view.

11.1.3 Notation

Throughout the paper we use m to denote the number of targets and n to denote the number of cameras. Thus there are $O(nm^2)$ pinned cones. We use variables i, j, k to refer to indices of cameras, targets, and cones, respectively. We sometimes write $c(k)$ to indicate the camera underlying a given cone k. At a given point in time while an algorithm runs, $B(k)$ indicates the set of targets that would be (re)assigned to the camera underlying cone k, if cone k were selected at that point.

11.2 Related Work

The pan and scan problem can be understood as a variant of the Maximum Coverage problem, a dual problem to Set Cover, in which, given a set system and an integer bound k, the goal is to choose at most k sets to cover a maximum-weight set of elements. The greedy algorithm provides a $1 - 1/e$ approximation, which is the best guarantee achievable unless

P=NP [38].

A variant of Maximum Coverage was studied in [30] the elements to be covered were unweighted and side constraints partitioned the available sets into groups from each of which at most one set could be chosen. They provide a 2-approximation greedy algorithm and state without details that $1 - 1/e$ is achievable through LP-rounding. Their version did not include element weights, and was non-geometric. Other related problems include the Multiple-Choice Knapsack problem, the Budgeted Maximum Coverage problem, and the Generalized Maximum Coverage problem [32].

One recent work motivated by similar applications is [19], in which directional antennas are configured in order to cover clients. The problems considered in that paper are similar in that the directional antenna can extend its range, distance-wise, by narrowing the angle at which it transmits. The specific optimization constraints and goals are quite different, however–limited the number of targets covered per antenna or minimizing the number of antennas activated, for example.

More broadly speaking, our problem is a geometric coverage problem of a similar ilk of the the one-dimensional and two-dimensional sensor radius problems (see [8] and references therein). Finally, measurement issues are dealt with from another, more physical perspective, in the statistics and engineering literatures (see [29] and references therein).

11.3 A Polynomial-time Solvable Constrained Setting

In this section, we consider a constrained special case which, unlike the general two-dimensional setting discussed below, can be solved optimally in polynomial time. In this setting, there is a *natural ordering* on the targets, the cameras are *interchangeable*, and there always exist optimal solutions that are *overlap-free*.

Definition 11.3.1. *We say a solution is* overlap-free *if no two of its pinned cones overlap unless one fully contains the other and that a setting is* overlap-free *if every instance admits an optimal solution which is overlap-free. Cameras are* interchangeable *if for each target set, all potential pinned cones covering it provide the same utility. The targets of an instance are* naturally ordered*, with ordering $j_1, ..., j_n$, if for any three targets j_1, j_2, j_3 a cone cannot capture j_1, j_3 without also capturing j_2, and there exist optimal solutions in which no cone captures j_n, j_1, but no other cones (unless $n = 2$).*

One example setting satisfying these properties is when cameras are *coextensive*, located at some point which is linearly separable from the target locations. But reduction to m separate naturally orderable subproblems, the setting of coextensive cameras and targets located anywhere in the plane can also be solved.

Algorithm 9 Overlap-free DP (for target sequence $(1, ..., m)$)

1: for all $0 \leq j_3 \leq j_1 \leq j_2 \leq j_4 \leq m + 1$, set $opt[j_1, j_2][j_3, j_4][0] \leftarrow 0$
2: for all $1 \leq j_1 \leq j_2 \leq m$ set $cv[j_1, j_2] \leftarrow$ the measurement value of the cone defined by j_1 and j_2
3: **for** $i = 1$ to m **do**
4: **for** for all $j_3 \leq j_1 \leq j_2 \leq j_4$ **do**
5: compute $opt[j_1, j_2][j_3, j_4][i]$ as the best among $opt[j_1 + 1, j_2 - 1][j_1, j_2][i - 1] + 2cv[j_1, j_2]$ and $\max_{i' \leq i} opt[j_1, j_0][j_3, j_4][i']opt[j_0 + 1, j_2][j_3, j_4][i - i']$
6: **end for**
7: **end for**
8: **return** $opt[1, m][0, m + 1][n]$

We solve this setting with a dynamic programming (DP) procedure (see Algorithm 9) which works recursively on the target sequence $1, ..., m$, computing optimal solutions for each combination an interval $[j_1, j_2]$, a superinterval $[j_3, j_4]$, and an integer i between 1 and n, i.e. filling in a table with entries of the form $opt[j_1, j_2][j_3, j_4][i]$. This subinstance corresponds to the the problem of covering interval $[j_1, j_2]$ with i cameras, plus the availability

of a camera already assigned to the larger interval $[j_3, j_4]$. For convenience, we assume the existence of a zero-utility camera assigned to the interval $[0, n+1]$, which contains all genuine intervals.

For each such problem subinstance, there are two kinds of cases to consider. Either one of the i cameras is used to cover both targets j_1 and j_2, or not. Assume so. Then we consider $opt[j_1+1, j_2-1][j_1, j_2][i-1]$. Now assume not. Then we consider every possible way to carve up the interval $[j_1, j_2]$ into two pieces, say $[j_1, j_0]$ and $[j_0+1, j_2]$, combined with every possible way to divvy up the i cones between the two pieces, that is, the combination of $opt[j_1, j_0][j_3, j_4][i']$ and $opt[j_0+1, j_2][j_3, j_4][i-i']$, for each $j_1 \leq j_0 < j_2$ and each $0 \leq i' \leq i$.

The total running time of the procedure is clearly polynomial.

Theorem 11.3.1. *For every naturally ordered, interchangeable-cameras, overlap-free setting, Algorithm 9 produces an optimal solution.*

Proof: (sketch) The proof of correctness is by induction on problem subinstances, over parameter i, the number of cameras available for use in the subinstance. Assuming optimal values are correctly computed for all subinstances in which the number of cameras is less than i, we correctly compute the optimal solution value for the current subinstance with i cameras, by construction.

Theorem 11.3.2. *Naturally ordered and interchangeable cameras implies overlap-free.*

Proof: Suppose that two pinned cones overlap without one containing the other, i.e., that one cone k_1 is defined by targets a, d and another k_2 defined by b, e, where $a < b < d < e$ (where the comparison operator indicates the order in which the targets appear on the circle). Suppose that lying between b and d are 0 or more targets $c_1, ..., c_r$, each assigned

to either k_1 or k_2. Among k_1 and k_2, let one have angle greater or equal to the other's, say k_2. Then (re)assign b,d, and each c_j to k_1, which will only increase the value of these reassigned targets, and shrink k_2 so that its coverage begins at the first target assigned to it following d, which will only increase the value of k_2's targets.

Corollary 11.3.3. *The setting with coextensive cameras, all located at a point* linearly separable *from the targets, is polynomial-time solvable.*

Proof: The natural ordering of the targets in this setting is the order in which a ray originating at the camera point, encounters the targets as it sweeps across the half-plane containing them.

Now consider a setting in which the cameras are coextensive but the targets can lie anywhere in the plane. Although this setting's optimal solutions are overlap-free, the targets are not given as a linear sequence. We therefore solve the circular setting by solving m all corresponding linear sequences.

Corollary 11.3.4. *The setting with coextensive cameras and targets located in the plane is polynomial-time solvable.*

Proof: We execute the DP procedure m times. In any optimal solution, there will be at least one pair of consecutive targets $j-1, j$ that are not both covered by any one cone. Therefore for each value j from 1 to m, we rename the target indices so that j is the first, proceeding clockwise, and $j-1$ is the last. We run the DP procedure on each such linear sequence, returning the best of these m solutions as the result.

11.4 Hardness

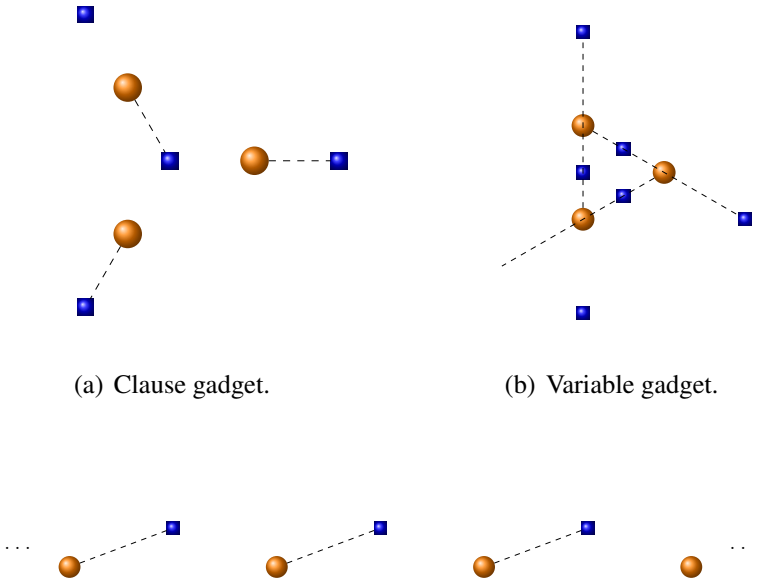
Let the two-dimensional camera configuration problem defined as in IP 11.1.2, except that focal length f_k^2 can be replaced (as discussed in Section 11.1.2) with any monotonic function $f(j,k)$ of field of view and $d(c(k),j)^2$ becomes any *faster growing* monotonic function of distance, in the sense that increased distance cannot be *entirely* made up for by zooming in, which is a realistic assumption.

We reduce from a restriction of Planar 3SAT [75] in which each variable appears in at most three clauses, which remains NP-hard [79]. Given the bipartite graph corresponding to such a 3SAT instance, we produce a problem instance. All locations are chosen in such a way that "adjacent" cameras and targets lie exactly unit distance apart, with the exception of the cameras and targets composing the variable gadgets, which are positioned closer together. Let u_1 be the utility provided by a camera maximally zoomed in on a target at distance 1 away; let $u_{1/4}$ be defined similarly except with distance $1/4$. We say that a variable target covered with utility $u_{1/4}$, or respectively a non-variable target covered with utility u_1, is covered *perfectly*. The problem instance is constructed so that given a positive 3SAT instance, in an optimal solution to the resulting instance, each target will be covered perfectly. In such a solution, each variable gadget will be oriented clockwise or counterclockwise, and each edge chain will either be oriented towards its clause or not.

For each clause we create a clause gadget as shown in Figure 11.2(a), in which a central target should be covered by one of three surrounding cameras, each located distance 1 away. Each camera is adjacent to a second target, which continues in an alternating chain of camera and target until reaching a variable gadget. (The chain can bend freely as long as the unit distance between adjacent nodes is maintained.) A variable gadget consists of three cameras arranged to form an equilateral triangle with side $1/2$ and three targets, each

group positioned to bisect one side of the triangle.

An edge gadget, corresponding to an appearance of this variable in the clause at the edge's other end, is attached to the variable by positioning the edge's final target collinear with one of the triangle's sides, at distance $1/2$ away. The dotted lines shown in Figure 11.2(c) are of unit length The target is positioned clockwise from its adjacent camera if the edge corresponds to a positive literal, and counter-clockwise otherwise. (The example in Figure 11.2(c) has two positive appearances and one negative.)



(a) Clause gadget.

(b) Variable gadget.

(c) Edge gadget.

Figure 11.2: Gadgets used in the reduction from Planar 3SAT.

We now claim the construction works correctly.

Lemma 11.4.1. *The 3SAT instance is satisfiable iff in an optimal solution every target is covered perfectly.*

Proof: First assume a solution with every target covered perfectly. Then each variable target must be covered by one of its two neighboring cameras. Due to the geometric nature of the variable gadget, deciding which camera will cover one target in a given variable determines which cameras will cover each of the other two targets, i.e., it determines an orientation of either clockwise or counter-clockwise for the variable. (The dotted lines shown in Figure 11.2(c) indicate a clockwise orientation.) The positioning of the edge chains as described above implies that a clockwise orientation will cover the final targets of adjacent positive edge chains, whereas a counter-clockwise orientation will cover the final targets of negative edge chains. In summary, each clause gadget must be covered perfectly, each edge chain is oriented, and each variable gadget has a consistent orientation. A satisfying assignment can then be read off from the variable orientations.

Now assume there is a satisfying assignment. For each clause, choose a variable appearing as a true literal in the clause under this assignment, and let the clause target be covered by the adjacent camera of the edge chain corresponding to this variable appearance. This choice orients the edge chain towards the clause. Orient the clause's other two chains away from the clause. Now consider the resulting state of some variable gadget. If any of its appearances was used to make a clause true, then some of its edge chains are oriented away from it, meaning that these chains' final targets are not covered. This determines an orientation for the variable gadget is chosen, either clockwise or counter-clockwise, in order to cover this target. A consistent orientation for each gadget is implied by the consistency of the satisfying assignment.

Thus we conclude:

Theorem 11.4.2. *The two-dimensional camera configuration problem is NP-hard.*

11.5 Algorithms

11.5.1 Combinatorial Algorithms

In the problem instance, we have a total of $O(nm^2)$ pinned cones to choose among. The Greedy algorithm (see Algorithm 10) will repeatedly select a pinned cone that increases the current solution value by the maximum amount, i.e., when all captured targets that would benefit, whether not yet assigned or already assigned but receiving lower utility, are assigned to the newly chosen cone. Once a given camera is assigned a cone, all other cones for that camera are removed from consideration.

Algorithm 10 Greedy

1: $C \leftarrow$ all possible pinned cones
2: for each target j and cone k, initialize $val(j,k)$ as the value of cone k measuring target j
3: **while** $C \neq \emptyset$ **do**
4: for each pinned cone k let $v_k = \sum_j val(j,k)$
5: select a most profitable pinned cone k, based at some camera i, and (re)assign each target j such that $val(j,k) > 0$ to k
6: for each remaining cone $k' \in C$ and for each target j now assigned to k, set $val(j,k') = \max\{val(j,k) - val(j,k'), 0\}$
7: remove from C all cones based at i and all cones of value zero
8: **end while**

Extending the analysis of [30], we first name the cones that Greedy chooses $g_1, ..., g_n$, in order. Next, we name the cones of an optimal solution OPT $h_1, ..., h_n$, arranged so that the cameras of these cones appear in the same order as the cameras of the cones chosen by Greedy. Let $A'_k = (a'_{1,k}, ..., a'_{m,k})$ be the vector of measurement quality increases for the m targets, as a result of Greedy choosing the kth cone, and let $O_k = (o_{1,k}, ..., o_{m,k})$ be the coverage values for the m targets by cone h_k. Let $ALG = (c_1, ..., c_m)$ and $OPT = (o_1, ..., o_m)$ be the final measurement qualities for the targets by Greedy and OPT, respectively.

Lemma 11.5.1. *For each k, $\sum_j a'_{j,k} \geq \sum_j \max\{o_{j,k} - c_j, 0\}$.*

Proof: (sketch; see also Lemma 11.6.1 below) If cone g_k is the cone for camera $c(k)$ in OPT, then $g_k = h_k$ and so the inequality holds, so assume otherwise. Then Greedy chose a cone g_k because the total increase A'_k was larger than the increase would have been by choosing cone o_k, which increase is larger still than the of total gains of o_k beyond the final values c_j.

Theorem 11.5.2. *Algorithm 10 provides a 2-approximation.*

Proof: Applying the lemma, we have:

$$\begin{aligned} ALG = \sum_k A'_k = \sum_{k,j} a'_{j,k} &\geq \sum_{j,k} \max\{o_{j,k} - c_j, 0\} \\ &\geq \sum_j \max\{\max_k o_k - c_j, 0\} \\ &\geq OPT - ALG \end{aligned}$$

Therefore $ALG \geq OPT/2$.

Let m be number of targets and n be number of cameras. Then in a naive implementation of Algorithm 10, the initialization step and each iteration of the loop takes time $O(nm^3)$, for a total of $O(n^2m^3)$. Using quality functions that degrade arbitrarily quickly with distance, it is easy to construct instances for which the approximation factor approaches 2.

Next we consider a distributed version of the greedy algorithm that assumes an asynchronous model in which cameras perform their cone-selection choices in arbitrary order, but we assume no two cameras will make their choices at precisely the same time.

Proposition 11.5.3. *Algorithm 11 provides a 2-approximation.*

Algorithm 11 Distributed Greedy

1: for each target j, initialize received quality $q_j \leftarrow 0$
2: **for** each camera c *in arbitrary order* **do**
3: for each target j and cone k of c, initialize $val(j,k)$ as the value of cone k measuring target j
4: for each pinned cone k let $v_k = \sum_j \max\{val(j,k) - q_j, 0\}$
5: select a most profitable pinned cone k based at camera c, and (re)assign each target j such that $val(j,k) > 0$ to k
6: for each target j, update $q_j \leftarrow q_j - val(j,k)$
7: **end for**

Proof: Observe that the lemma still obtains because when camera i chooses cone k, it does so because the benefit is greater or equal to choosing whatever cone OPT contains for i. The proof then goes through unchanged.

Each iteration of the loop completes in time $O(m^3)$, for a total of $O(nm^3)$ time in the naive implementation.

Although in principle we allow for cameras observing targets at arbitrarily large distances, we might reasonably truncate the measurement quality to 0 for all targets at distance beyond some threshold. In this case, when running the distributed algorithm, a camera need only keep track of the assignment state of all targets that are *local* in this sense.

11.5.2 Efficient Implementation

We now explain how to implement the subroutine of choosing the best (current) cone for a given camera in time $O(m^2)$, on the assumption that the measurement quality for a cone/target pair is a multiplicative function of distance and zoom factor.

Fix a particular camera i, and renumber the targets 0 to $m-1$ as they are encountered by a ray originating at i (the *hand*), sweeping around clockwise. We we proceed in m rounds, computing the values of n cones each time. In round j_1, we compute the values

of all cones picked out by targets j_1 and j_2 as j_2 varies (wrapping around modulo m) from $j_1 + 1$ to j_1, with the first cone observing all targets and the last observing only target j_2. Each new cone value is computed in constant time from the previous one, for a total of $O(m)$ per round.

Before beginning the rounds, we do some initial computation involving targets in order to support the cone evaluation. Let q_j be the current quality value of the targets j, and let $val(j,k) = 1/d_{j,k}f_k$ be the value with which cone k can read target j, which is the product of the distance component and the product component. The value of cone k will be the sum of the *potential positive increases* of these values, i.e., the sum of $D_k = \Delta_{j,k} = \max\{val(j,k) - q_k, 0\}$, summed over all the cone's targets j, which is the same as the sum of the values $val(j,k) - q_k$, restricted to the cone's targets *that the cone would benefit*. Call this set of targets B_k. This can be rewritten as $f_k \sum_{j \in B_k} 1/d_{j,k} - \sum_{j \in B_k} q_k = f_k D_k - q_j$

Each time the hand moves from one target to the next, one target is removed from evaluation, which means removing its $\Delta_{j,k}$. On the other hand, the value of the remaining targets increases, since the angle is now smaller. As the hand rotates around the camera, more and more targets depart from the cone. We keep stored the current value of Q_k and D_k. To compensate for the higher quality readings of the remaining targets, after removing each lost target, it suffices to update f_k, D_k, and Q_k. Updating f_k is done in constant time. To update the other two values, we must add the corresponding sums restricted to the *targets newly entering* B_k.

But which targets enter B_k each time the hand moves? A target is in B_k exactly when $f_k/d_{j,k} - q_j > 0$, i.e., when $f_k > q_j d_{j,k} = T_j$, where we call T_j the threshold for target j. (Note that within the computation for a single camera, q_j and $d_{j,k}$ are constants.) We can compute this and sort the values T_j in time $O(m \log m)$. Now consider the beginning

of any particular round. All targets are contained in the cone. The members of B_k are all targets for whom currently $T_j > f_k$. We maintain a pointer to the first target for which this is not so. Each time the hand advances, f_k increases and we advance the pointer, adding targets to B_k, and updating Q_k and D_k until we reach a target whose threshold is not met. It therefore takes $O(m)$ time to update the Q_k and D_k values in each round, for a total of $O(m^2)$ time.

Thus we conclude the following.

Theorem 11.5.4. *Let measurement quality be the product of functions of distance and zoom factor. Then Algorithms 10 and 11 can be implemented to run in times $O(n^2m^2)$ and $O(nm^2)$, respectively.*

11.5.3 A PTAS

We note that with the assumption of bounded measurement distance, plus the assumption that the graph of cameras and targets is *drawn in a civilized manner* [61] (with locations specified at finite precision and no two cameras or targets occupying exactly the same spot), a grid-shifting PTAS [57] can be obtained. We briefly sketch the details.

First, the existence of a global constant bounding the distance at which measurements of nonzero value are possible provides some locality, so for the targets within any enclosed area, any camera they are assigned to must lie sufficiently nearby. Second, the civilized-manner assumption means that within any region of constant area the number of cameras and targets will be bounded by a constant, and so the problem subinstance corresponding to that area can be solved in constant time. Together, these two assumptions allow us to implement a grid-shifting PTAS in which a grid, of precision based on the desired approximation guarantee, is lain on the plane. For each possible offset, we solve the problem

subinstance contained within each cell of the grid, disregarding target-camera edges that cross the grid lines, unioning the results together. The best solution over all possible offsets is the solution returned. The grid is chosen to be sufficiently coarse so that the value of the optimal solution disregarding crossedges will be a good enough approximation, for at least one possible grid offset.

11.6 Distributed Synchronous Protocols

As noted above, Algorithm 11 is a distributed algorithm in the sense that each camera can choose its configuration independently of other cameras, assuming no conflicts, i.e., assuming that each camera c's decision is made quickly enough so that while c is interrogating the sensors within its range, no other camera covers them simultaneously.

In this section, we discuss more robust distributed protocols that maintain approximation guarantees but relax the no-collision assumption.

Algorithm 12 Rounds Protocol

1: $C \leftarrow \{1, ..., n\}$
2: **while** $C \neq \emptyset$ **do**
3: for each camera $i \in C$, let $c(i)$ be a best potential cone for i, based on $B(i)$, which is the set of targets lying inside $c(i)$ to which $c(i)$ would provide an increase in coverage quality
4: each camera $i \in C$ sends a coverage proposal to all targets $j \in B(i)$
5: each target that receives at least one proposal accepts the proposal coming from the camera of lowest index number
6: each camera i that receives positive responses from *all* the targets it proposes to chooses cone $c(i)$ and has all targets in $B(i)$ assigned to it
7: **end while**

Algorithm 12 can be understood as a synchronous, rounds-based implementation of

Algorithm 11 above, and so the approximation guarantee remains unchanged. Unfortunately, a given target may be assigned as many as n times over the course of the algorithm's execution. The next algorithm is designed limit the number of reassignments while compromising the approximation guarantee only slightly. Note that with parameter values $\alpha = 0, \beta = 1$, its behavior is the same as Algorithm 12's.

Algorithm 13 Bounded Reassignment Protocol (parameters $\alpha \geq 0, \beta \geq 1$)

1: $C \leftarrow \{1, ..., n\}$
2: for each camera $i \in C$, compute the best value $e(i)$ of a cone based at i observing a *single* target
3: perform a *leader election* protocol [44] in which the cameras broadcast their values $e(i)$ and all cameras learn $e_{max} = max_i\{e(i)\}$
4: **while** $C \neq \emptyset$ **do**
5: for each camera $i \in C$, let $c(i)$ be a best potential cone for i, based on $B(i)$, which is the set of targets lying inside $c(i)$ to which $c(i)$ would provide a coverage quality which is at least $\alpha \cdot e_{max}/m$ and is at least β times the target's current coverage value
6: each camera $i \in C$ sends a coverage proposal to all targets $j \in B(i)$
7: each target that receives at least one proposal accepts the proposal coming from the camera of lowest index number
8: each camera i that receives positive responses from *all* the targets it proposes to chooses cone $c(i)$ and has all targets in $B(i)$ assigned to it
9: **end while**

The following lemma is a generalization of Lemma 11.5.1 above.

Lemma 11.6.1. *Let $\alpha = 0$ but allow $\beta \geq 1$. Then for each k, $\sum_j a'_{j,k} \geq \sum_j \max\{\frac{1}{\beta} o_{j,k} - c_j, 0\}$.*

Proof: If the algorithm chooses the same cone for camera k as OPT does, then for each target j that o_k covers, target j is not assigned to camera k only if it is already covered with value greater than $1/\beta$ the value it could receive from camera k. On the other hand, if the algorithm chooses a different cone for camera k, then this is because its choice of cone seemed sufficiently good in the following sense. Let $val(j)$ be the current coverage value

for target j at the time at which camera k makes its choice. Let $a'_{j,k}$, $a_{j,k}$, $o_{j,k}$, and c_j be defined as in Section 11.5, with the understanding that a target j is assigned to cone k, and hence contributes to $a'_{j,k}$, only if it the value passes the tests based on α and β. Then the total increase due to ALG's choice of cone for camera k is

$$\begin{aligned}\sum_j a'_{j,k} = \sum_j \max\{a_{j,k} - val(j), 0\} &\geq \sum_j \max\{1/\beta o_{j,k} - val(j), 0\} \\ &\geq \sum_j \max\{1/\beta o_{j,k} - c_j, 0\}\end{aligned}$$

Theorem 11.6.2. *Algorithm 13 provides a* $(2\beta)/(1-\alpha)$*-approximation guarantee.*

Proof: Since at least one target can receive coverage value e_{max} but no target can receive value greater than this, we have that $e_{max} \leq OPT \leq m \cdot e_{max}$. Suppose $\alpha = 0$ but possibly $\beta > 0$. Then by applying the lemma, we can expand the value of ALG as before yields the result that $ALG \geq OPT/\beta - ALG$, and so $ALG \geq OPT/(2\beta)$.

Now suppose $\alpha > 0$. The behavior of Algorithm 13 is unchanged if we eliminate all edges of value less than $\alpha \cdot e_{max}/m$. Since each target gets coverage quality from only one camera, the optimal solution value decreases by at most $\sum_j \alpha \cdot e_{max}/m = \alpha \cdot e_{max} \leq \alpha OPT$. That is, the optimal solution value of the modified instance OPT' is at least $(1-\alpha)OPT$. Applying the previous approximation guarantee to the modified instance, we conclude with a guarantee of $(1-\alpha)OPT/(2\beta)$.

Proposition 11.6.3. *Algorithm 13 reassigns each target at most* $\log_\beta(m/\alpha)$ *times, where* m *is the number of targets.*

Proof: In the course of the algorithm, after initially being covered, a target's coverage quality varies from a minimum of $\alpha e_{max}/m$ to a maximum of e_{max}, each time

growing by a factor of at least β. The total number of reassignments is therefore at most $\log_\beta(e_{max}/(\alpha e_{max}/m)) = \log_\beta(m/\alpha)$.

In the worst case, Algorithms 12 and 13 can take n rounds to complete. For graphs will bounded degrees, however, fewer rounds are required.

Proposition 11.6.4. *In the bipartite graph of cameras and targets, where edges appear if the camera can potentially provide nonzero utility to the target, assume the degrees of the camera nodes and target nodes are bounded by constants Δ_c and Δ_t, respectively. Then Algorithms 12 and 13 both complete within $\log_{\Delta_c(\Delta_t-1)}(n)$ rounds.*

Proof: Consider a camera i sending proposal in a given round. It sends proposals to at most Δ_c targets, each of which receives proposals from at most Δ_t cameras. Suppose camera i receives its cone in this round. In the worst case, c interferes with $\Delta_c(\Delta_t - 1)$ other cameras. More generally, in the worst case only one in $\Delta_c(\Delta_t-1)$ remaining cameras receives a cone in each round.

Using the speed-up techniques of Section 11.5.2, we conclude with the following.

Proposition 11.6.5. *A camera's work within each round can be implemented to run in time $O(n^2)$ or, given degree bound Δ_c, time $O(\Delta_c^2)$.*

11.7 Open Problems

One clear open problem is to find a PTAS based on weaker assumptions, e.g. not relying on the civilized drawing assumption, or to prove its nonexistence.

Another is to find polynomial-time optimal algorithms, based e.g. on dynamic programming, for one-dimensional problem settings. In one natural one-dimensional setting, with

cameras and targets appearing on the line, when a camera chooses some target to zoom in on, it observes that target and all subsequent targets, albeit with lower measurement quality.

More conceptually, the camera configuration problem could be generalized in two natural ways, aggregation functions and utility functions. We studied aggregation functions SUM and MAX, but other aggregation functions could be considered, including functions that may be tuned to trade-off between false-positive and false-negative error (see [29]). Orthogonally, in some applications it might make sense to enforce limits (other than 1 or n) on the number of readings allowable for a given target or on the number of targets observable within a single cone.

Bibliography

[1] Gaia Power Technologies. gaiapowertech.com.

[2] NEOS server, Argonne National Lab. www-neos.mcs.anl.gov/neos/solvers/.

[3] ConEd electricity rates document. www.coned.com/documents/elec/043-059h.pdf.

[4] Orlando Utilities Commission website. www.ouc.com/account/rates/electric-comm.htm.

[5] S. Aaronson. NP-complete problems and physical reality. *Electronic Colloquium on Computational Complexity (ECCC)*, (026), 2005.

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

[7] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows*. Prentice Hall, 1993.

[8] H. Alt, E. M. Arkin, H. Brönnimann, J. Erickson, S. P. Fekete, C. Knauer, J. Lenchner, J. S. B. Mitchell, and K. Whittlesey. Minimum-cost coverage of point sets by disks. In *Symposium on Computational Geometry*, pages 449–458, 2006.

[9] A. Atamturk and D. S. Hochbaum. Capacity acquisition, subcontracting, and lot sizing. *Management Science*, Vol. 47, No. 8, 2001.

[10] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 1991.

[11] X. Bai, S. Kumar, D. Xuan, Z. Yun, and T.-H. Lai. Deploying wireless sensors to achieve both coverage and connectivity. In *MobiHoc 2006*.

[12] A. Bar-Noy, T. Brown, M. P. Johnson, T. La Porta, O. Liu, and H. Rowaihy. Assigning sensors to missions with demands. In *ALGOSENSORS 2007*.

[13] A. Bar-Noy, T. Brown, M.P. Johnson, and O. Liu. Cheap or flexible sensor coverage. In *DCOSS 2009*.

[14] A. Bar-Noy, Y. Feng, M. P. Johnson, and O. Liu. When to reap and when to sow: Lowering peak usage with realistic batteries. In *WEA 2008*.

[15] A. Bar-Noy, M. Johnson, and O. Liu. Peak shaving through resource buffering. Technical Report TR-2007018, CUNY Graduate Center, Dept. of Computer Science, November 2007.

[16] A. Bar-Noy, M. P. Johnson, and O. Liu. Peak shaving through resource buffering. In *WAOA*, pages 147–159, 2008.

[17] N. Bartolini, T. Calamoneri, E. G. Fusco, A. Massini, and S. Silvestri. Snap and spread: A self-deployment algorithm for mobile sensor networks. In *DCOSS 2008*.

[18] P. Berman. A d/2 approximation for maximum weight independent set in d-claw free graphs. In *Proceedings of SWAT*, pages 214–219, 2000.

[19] P. Berman, J. K. Jeong, S. P. Kasiviswanathan, and B. Urgaonkar. Packing to angles and sectors. In *SPAA*, pages 171–180, 2007.

[20] M. Bernstein, N. J. A. Sloane, and P. E. Wright. On sublattices of the hexagonal lattice. *Discrete Mathematics*, 170:29–39, 1997.

[21] D. Bertsekas and R. Gallager. *Data Networks, 2nd Edition*. Prentice Hall, 1991.

[22] V. Bilò, I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. Geometric clustering to minimize the sum of cluster sizes. In *ESA*, 2005.

[23] C. Bisdikian. On sensor sampling and quality of information: A starting point. In *3rd IEEE Int'l Workshop on Sensor Networks and Systems for Pervasive Computing (PerSeNS 2007), a PerCom'07 Workshop*, White Plains, NY, March 19-23, 2007.

[24] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[25] P. Brass. Bounds on coverage and target detection capabilities for models of networks of mobile sensors. *ACM T. on Sensor Networks*, 3(2), 2007.

[26] P. Brass, W. O. J. Moser, and J. Pach. *Research Problems in Discrete Geometry*. Springer, Berlin-Heidelberg, 2005.

[27] T. Brown, D. Sarıöz, A. Bar-Noy, T. La Porta, D. Verma, M.P. Johnson, and H. Rowaihy. Geometric considerations for distribution of sensors in ad-hoc sensor networks. In *SPIE DSS 2007*.

[28] J. Byers and G. Nasser. Utility-based decision-making in wireless sensor networks. In *Proceedings of MobiHoc '00*, 2000.

[29] Z. Charbiwala, Y. Kim, S. Zahedi, J. Friedman, and M. B. Srivastava. Energy efficient sampling for event detection in wireless sensor networks. In *ISLPED*, pages 419–424, 2009.

[30] C. Chekuri and A. Kumar. Maximum coverage problem with group budget constraints and applications. In *APPROX-RANDOM*, pages 72–83, 2004.

[31] B. Clark, C. Colbourn, and D. Johnson. Unit disk graphs. *Discrete Math*, 86:165–177, 1990.

[32] R. Cohen and L. Katzir. The generalized maximum coverage problem. *Inf. Process. Lett.*, 108(1):15–22, 2008.

[33] R. Cohen, L. Katzir, and D. Raz. An efficient approximation for the generalized assignment problem. *Inf. Process. Lett.*, 100(4):162–166, 2006.

[34] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill, 2001.

[35] S. de Vries and R. Vohra. Combinatorial auctions: a survey. *INFORMS J. on Computing*, 15-3:284–309, 2003.

[36] T. Erlebach and J. Fiala. Independence and coloring problems on intersection graphs of disks. *manuscript*, 2001.

[37] T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation schemes for geometric graphs. In *SODA*, 2001.

[38] U. Feige. A threshold of ln for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[39] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *SODA 06*, pages 611–620, 2006.

[40] R. Fleischer. On the bahncard problem. *Theoretical Computer Science*, Vol. 268(Issue 1):161–174, October 2001.

[41] M. Florian, J. Lenstra, and A. R. Kan. Deterministic production planning: algorithms and complexity. *Management Science*, Vol. 26, 1980.

[42] S. Funke, A. Kesselman, F. Kuhn, Z. Lotker, and M. Segal. Improved approximation algorithms for connected sensor cover. *Wireless networks*, 13(2):153–164, 2007.

[43] Z. Galil, S. Micali, and H. Gabow. An O(EV log V) algorithm for finding a maximal weighted matching in general graphs. *SIAM J. Comput.*, 15(1):120–130, 1986.

[44] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

[45] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Colmpleteness*. Freeman, 1979.

[46] M. Gibson, G. Kanade, E. Krohn, I. Pirwani, and K. Varadarajan. On clustering to minimize the sum of radii. In *SODA*, 2008.

[47] M. Goh, O. Jihong, and T. Chung-Piaw. Warehouse sizing to minimize inventory and storage costs. *Naval Research Logistics*, Vol. 48, Issue 4, 3 Apr 2001.

[48] N. Goodman. *Fact, Fiction, and Forecast, Fourth Edition*. Harvard University Press, 2006 (originally published 1954).

[49] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science, 2nd Edition*. Addison-Wesley Professional, 1994.

[50] H. Gupta, Z. Zhou, S. R. Das, and Q. Gu. Connected sensor cover: self-organization of sensor networks for efficient query execution. *IEEE/ACM Transactions on Networks*, 14:55–67, 2006.

[51] D. P. Hardin and E. B. Saff. Discretizing manifolds via minimum energy points. *Notices of the AMS*, 51(10):1186–1194, 2004.

[52] T. Harford. The great Xbox shortage of 2005. *Slate*, Dec. 15, 2005 www.slate.com/id/2132071/.

[53] J. Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[54] E. Hazan, S. Safra, and O. Schwartz. On the complexity of approximating k-set packing. *Computational Complexity*, 15(1):20–39, 2006.

[55] M. Hefeeda and H. Ahmadi. A probabilistic coverage protocol for wireless sensor networks. In *Proc. of IEEE International Conference on Network Protocols (ICNP'07)*, Beijing, China, October 2007.

[56] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocols for wireless microsensor networks. In *Proc. Hawaaian Int'l Conf. on Systems Science*, 2000.

[57] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32(1):130–136, 1985.

[58] C.-F. Huang and Y.-C. Tseng. The coverage problem in a wireless sensor network. *Mobile Networks and Applications*, 10(4):519–528, 2005.

[59] D. Hume. *An Enquiry concerning Human Understanding*. Oxford University Press, 2001 (originally published 1777).

[60] B. Hunsaker, A. J. Kleywegt, M. W. P. Savelsbergh, and C. A. Tovey. Optimal online algorithms for minimax resource scheduling. *SIAM J. Discrete Math.*, 2003.

[61] H. Hunt, M. Marathe, V. Radhakrishnan, S. Ravi, D. Rosenkrantz, and R. Stearns. NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. *J. Algorithms*, 26(2):238–274, 1998.

[62] A. Iosevich, E. Sawyer, and A. Seeger. Mean square discrepancy bounds for the number of lattice points in large convex bodies. *Journal of the Annals of Mathematics*, 87:209–230, 2002.

[63] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *Journal of the ACM*, 50(6):795–824, 2003.

[64] L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. *Distributed Computing*, 15(4):193–205, 2002.

[65] M. P. Johnson, H. Rowaihy, D. Pizzocaro, A. Bar-Noy, S. Chalmers, T. La Porta, and A. Preece. Frugal sensor assignment. In *DCOSS '08*, 2008.

[66] M. P. Johnson, D. Sarioz, A. Bar-Noy, T. Brown, C. W. Wu, and D. Verma. More is more: the benefits of dense sensor assignment. In *INFOCOM 2009*.

[67] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava. Power management in energy harvesting sensor networks. *ACM Trans. Embedded Comput. Syst.*, (6(2)), 2007.

[68] L. Kaplan. Global node selection for localization in a distributed sensor network. *IEEE Transactions on Aerospace and Electronic Systems*, 42(1):113–135, January 2006.

[69] B. Karp and H. Kung. Greedy perimeter stateless routing for wireless networks. In *Proceedings of MobiCom '00*, pages 243–254, Boston, MA, August 2000.

[70] R. Karp, U. Vazirani, and V. Vazirani. An optimal algorithm for online bipartite matching. In *Proceedings of STOC*, 1990.

[71] R. Kershner. The number of circles covering a set. *Amer. J. Math.*, 61:665–671, 1939.

[72] M.-K. Lee and E. A. Elsayed. Optimization of warehouse storage capacity under a dedicated storage policy. *Int J Prod Res*, Vol. 43, No. 9, 2005.

[73] J. Lenchner. A faster dynamic programming algorithm for facility location. In *FWCG*, 2006.

[74] N. Lev-Tov and D. Peleg. Polynomial time approximation schemes for base station coverage with minimum total radii. *Computer Networks*, 47(4):489–501, 2005.

[75] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.

[76] J. Lu, L. Bao, and T. Suda. Coverage-aware sensor engagement in dense sensor networks. In *Proceedings of the International Confernece on Embedded and Ubiquitous Computing - EUC 2005*, Dec. 2005.

[77] M. Marathe, V. Radhakrishnan, H. Hunt, and S. Ravi. Hierarchically specified unit disk graphs. *Theor. Comput. Sci.*, 174(1-2):23–65, 1997.

[78] T. Matsui. Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs. In *Proceedings of the Japan Conference on Discrete and Computational Geometry*, pages 194–200, 1998.

[79] M. Middendorf and F. Pfeiffer. On the complexity of the disjoint paths problem. *Combinatorica*, (13(1)):97–107, 1993.

[80] T. Mullen, V. Avasarala, and D. L. Hall. Customer-driven sensor management. *IEEE Intelligent Systems*, 21(2):41–49, Mar/April 2006.

[81] J. Pach and P. Agarwal. *Combinatorial Geometry*. Wiley-Interscience, New York, 3rd edition, 1995.

[82] J. Pach and G. Tóth. Decomposition of multiple coverings into many parts. In *Symposium on Computational Geometry*, pages 133–137, 2007.

[83] M. Perillo and W. Heinzelman. Optimal sensor management under energy and reliability constraints. In *Proceedings of the IEEE Conference on Wireless Communications and Networking*, March 2003.

[84] Y. Pochet and L. Wolsey. *Production Planning by Mixed Integer Programming*. Springer, 2006.

[85] S. Poduri and G. S. Sukhatme. Constrained coverage for mobile sensor networks. In *ICRA 2004*, pages 165–171.

[86] H. Rowaihy, S. Eswaran, M. P. Johnson, D. Verma, A. Bar-Noy, T. Brown, and T. La Porta. A survey of sensor selection schemes in wireless sensor networks. In *SPIE Defense and Security Symposium*, 2007.

[87] H. Rowaihy, M. P. Johnson, A. Bar-Noy, T. Brown, and T. F. L. Porta. Assigning sensors to competing missions. In *GLOBECOM*, pages 44–49, 2008.

[88] H. Rowaihy, M. P. Johnson, T. Brown, A. Bar-Noy, and T. La Porta. Assigning Sensors to Competing Missions. Technical Report NAS-TR-0080-2007, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, October 2007.

[89] H. Rowaihy, M. P. Johnson, D. Pizzocaro, A. Bar-Noy, L. Kaplan, T. La Porta, and A. Preece. Detection and localization sensor assignment with exact and fuzzy locations. In *DCOSS 2009*.

[90] K. Shih, Y. Chen, C. Chiang, and B. Liu. A distributed active sensor selection scheme for wireless sensor networks. In *Proceedings of the IEEE Symposium on Computers and Communications*, June 2006.

[91] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, (28):202–208, 1985.

[92] K. E. Spaulding, R. L. Miller, and J. Schildkraut. Methods for generating blue-noise dither matrices for digital halftoning. *Journal of Electronic Imaging*, 6(2):208–230, 1997.

[93] S. C. Sung and M. Vlach. Maximizing weighted number of just-in-time jobs on unrelated parallel machines. *J. Scheduling*, 8-5:453–460, 2005.

[94] R. Ulichney. *Digital Halftoning*. MIT Press, 3rd edition, 1997.

[95] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[96] H. Wagner and T. Whitin. Dynamic version of the economic lot size model. *Management Science*, Vol. 5, 1958.

[97] M. L. Wald. Storing sunshine. *The New York Times*, July 16, 2007 www.nytimes.com/2007/07/16/business/16storage.html.

[98] L. Wang and Y. Xiao. A survey of energy-efficient scheduling mechanisms in sensor networks. *Mobile Networks and Applications*, 11(5):723–740, 2006.

[99] Y.-C. Wang and Y.-C. Tseng. Distributed deployment schemes for mobile wireless sensor networks to ensure multilevel coverage. *IEEE Transactions on Parallel and Distributed Systems*, 19, September 2008.

[100] R. Womersley. Visualization of minimum energy points on the torus. `http://web.maths.unsw.edu.au/~rsw/Torus/`.

[101] C. W. Wu, G. Thompson, and M. Stanich. A unified framework for digital halftoning and dither mask construction: variations on a theme and implementation issues. In *Proceedings IS&T's NIP19: International Conference on Digital Printing Technologies*, pages 793–796, 2003.

[102] C. W. Wu and D. Verma. A sensor placement algorithm for redundant covering based on riesz energy minimization. In *Proceedings of IEEE ISCAS*, pages 2074–2077, 2008.

[103] H. Zhang and J. C. Hou. Is deterministic deployment worse than random deployment for wireless sensor networks? In *INFOCOM 2006*.

[104] H. Zhang and J. C. Hou. Maintaining sensing coverage and connectivity in large sensor networks. *Ad Hoc and Sensor Wireless Networks*, 1(1-2):89–124, 2005.

[105] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration. *IEEE Signal Processing Magazine*, 19(2):61–72, March 2002.

[106] Y.-W. Zhou. A multi-warehouse inventory model for items with time-varying demand and shortages. *Computers and Operations Research*, Vol. 30, Issue 14, December 2003.

[107] Z. Zhou, S. Das, and H. Gupta. Connected K-coverage problem in sensor networks. In *Proceedings 13th International Conference on Computer Communications and Networks*, pages 373–378, 2004.

[108] Y. Zou and K. Chakrabarty. Uncertainty-aware and coverage-oriented deployment for sensor networks. *Journal of Parallel and Distributed Computing*, 64(7):788–798, 2004.