SANDIA REPORT

SAND2016-9266
Unlimited Release
Printed September 2016

Dynamic Multi-Sensor Multi-Mission Optimal Planning Tool

Christopher G. Valicka, Stephen Rowe, Simon X. Zou, Scott A. Mitchell, William R. Irelan, Eric L. Pollard, Deanna Garcia, Gabriel Hackebeil, Andrea Staid, Mark D. Rintoul, Jean-Paul Watson, William E. Hart, Sivakumar Rathinam, Lewis Ntaimo

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2016-9266
Unlimited Release
Printed September 2016

Dynamic Multi-Sensor Multi-Mission Optimal Planning Tool

Christopher G. Valicka
Systems Mission Engineering

William R. Irelan
Systems Mission Engineering

Stephen Rowe
Systems Mission Engineering

Simon X. Zou
Systems Mission Engineering

Eric L. Pollard
Systems Mission Engineering

Deanna Garcia
Center for Computing Research

Gabriel Hackebeil
Center for Computing Research

Andrea Staid
Center for Computing Research

Scott A. Mitchell
Center for Computing Research

Mark D. Rintoul
Center for Computing Research

Jean-Paul Watson
Center for Computing Research

William E. Hart
Center for Computing Research

Sivakumar Rathinam
Texas A&M University, Mechanical Engineering

Lewis Ntaimo
Texas A&M University, Industrial and Systems Engineering

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1243
cgvalic@sandia.gov

Abstract

Remote sensing systems have firmly established a role in providing immense value to commercial industry, scientific exploration, and national security. Continued maturation of sensing technology has reduced the cost of deploying highly-capable sensors while at the same time increased reliance on the information these sensors can provide. The demand for time on these sensors is unlikely to diminish. Coordination of next-generation sensor systems, larger constellations of satellites, unmanned aerial vehicles, ground telescopes, etc. is prohibitively complex for existing heuristics-based scheduling techniques. The project was a two-year collaboration spanning multiple Sandia centers and included a partnership with Texas A&M University. We have developed algorithms and software for collection scheduling, remote sensor field-of-view pointing models, and bandwidth-constrained prioritization of sensor data. Our approach followed best practices from the operations research and computational geometry communities. These models provide several advantages over state of the art techniques. In particular, our approach is more flexible compared to heuristics that tightly couple models and solution techniques. First, our mixed-integer linear models afford a rigorous analysis so that sensor planners can quantitatively describe a schedule relative to the best possible. Optimal or near-optimal schedules can be produced with commercial solvers in operational run-times. These models can be modified and extended to incorporate different scheduling and resource constraints and objective function definitions. Further, we have extended these models to proactively schedule sensors under weather and *ad hoc* collection uncertainty. This approach stands in contrast to existing deterministic schedulers which assume a single future weather or *ad hoc* collection scenario. The field-of-view pointing algorithm produces a mosaic with the fewest number of images required to fully cover a region of interest. The bandwidth-constrained algorithms find the highest priority information that can be transmitted. All of these are based on mixed-integer linear programs so that, in the future, collection scheduling, field-of-view, and bandwidth prioritization can be combined into a single problem. Experiments conducted using the developed models, commercial solvers, and benchmark datasets have demonstrated that proactively scheduling against uncertainty regularly and significantly outperforms deterministic schedulers.

Acknowledgement

We would like to acknowledge John T. Feddema, Brian N. Post, John H. Ganter, and Swaroop Darbha for providing critical project stewardship and fruitful remote sensing utilization discussions. A special thanks to Mohamed S. Ebeida for his contributions to the development of the Maximal Poisson Sampling technique. We would also like to thank Kaarthik Sundar and Jianglei Qin for their significant scheduling algorithm and model development contributions to the project. The authors would like to acknowledge the Sandia LDRD program for their support of this work. Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Contents

Appendix

List of Figures

Executive Summary

The Dynamic Multi-Sensor Multi-Mission Optimal Planning Tool project investigated sensor scheduling problems over the course of fiscal years 2015 and 2016. This report is a culmination of work conducted by Sandians from the System Missions Engineering Center, the Center for Computing Research, and Texas A&M University. Separate publications resulting from this work are cited but not reproduced herein to keep the report concise.

Sensor scheduling problems were studied using tools and techniques from operations research and computational geometry. Multiple sets of models were formulated using these techniques, including set cover models for collection scheduling and geometric coverage models for boresite (sensor footprint) and chip (sensor sub-footprint) placement optimization. Both sets of models utilize the Sandia open-source optimization modeling language Pyomo© to instantiate flexible mixed-integer linear models. By making use of Pyomo, these models can be solved with commercial, open-source, and custom solvers.

Solution quality and runtime were both important considerations in designing the models and algorithms. By developing mixed-integer linear models, a linear relaxation of the problem can be solved quickly to provide a bound on the best possible schedule or placement solution. This stands in contrast with purely heuristic solvers that provide solutions quickly but without a quantitative comparison to the best possible schedule or placement. Using the project-developed models, sensor planners and operators can choose from a variety of solution methodologies depending on the size of the problem and the operational time-constraints for producing a usable solution. For example, small optimality gaps can be allowed so that off-the-shelf solvers can quickly produce solutions without the need for sophisticated tuning. Project-developed heuristics for both the collection scheduling and sensor sub-footprint placement problems can be quantitatively compared to other solution methods while still providing near real-time solutions.

Several other significant contributions were made to the sensor scheduling community. Specifically, stochastic mixed-integer linear programs were developed for the collection scheduling problem to proactively schedule against weather and *ad hoc* collection uncertainty. These models have demonstrated significant improvements in expected schedule utility over deterministic schedulers. Use of stochastic models stands as a promising approach for scheduling strategies that build upon tipping and cueing to achieve a notion of synthetic sensor persistence. Benchmark data instances were produced for use with the deterministic and stochastic mixed-integer linear models and serve as a key component for the results reported in Valicka et al. [43]. Source code for all collection scheduling models and heuristics are available to Sandians. Additional software was developed under the project including the open-source GeoPlace© software and an internally available web application for collection scheduling and schedule analysis.

Chapter 1

Constellation Collection Scheduling

1.1 Sensor Scheduling Problem Background

Remote sensing technology was historically developed for use primarily in scientific and military applications, deployed on government-owned and operated satellites and aircraft. While reliance on and development of remote sensing technology for these traditional applications continues, commercial satellite and airborne remote sensing systems are being increasingly deployed to support industrial applications. This shift is driven by rapid decreases in the cost of launching into orbit or manning aircraft and overall technical risk, particularly when assessed relative to the competitive advantage such systems provide. Yet, despite these trends, the cost of deploying and operating remote sensing technology is sufficiently high to ensure that the resources are continually over-subscribed, i.e., the number of collection requests far outpaces available sensor time. This is particularly true for satellite-based sensors designed for collecting observations of Earth, as compared to communication satellites. To help alleviate this bottleneck, companies like DigitalGlobe offer services to collect imagery for the purpose of assisting governments across the globe in maintaining maps and planning for infrastructure. However, even with commercial deployments, the demand for Earth observations is expected to continue to outpace the supply of available sensor time across the scientific, commercial, and defense spectrum.

Collection scheduling for sensors on orbital platforms following a variety of disparate trajectories is challenging in large part due to their ability to observe a diverse range of geospatial locations. Typically, a very large number of competing collection requests – almost universally more than can be accommodated – are considered when building a collection schedule. Any resulting schedule must account for the utility of each scheduled collection and constraints on when, where, and how the collection can be accomplished. For an individual Earth observing satellite, the number of potential sensor schedules is typically extremely large. For a constellation of two or more satellites, the number of feasible schedules can grow even larger, due to multiple sensors being able to satisfy a specific collection request. The number of potential collection schedules further depends on the time horizon over which a collection schedule is being built (e.g., 12 hours or one day) and the time granularity considered (e.g., one second or one minute).

Collection scheduling problems for constellations of Earth observing satellites are closely related to multiple machine, multiple job scheduling problems widely studied in the operations research and optimization literature [46, 2]. In job scheduling problems, one or more machines are

required to complete a number of jobs, each of which specifies a duration and target start and end times. The optimization objective is to complete the job set in the minimal amount of time required, while adhering to the job specifications and limits on machine resources. Formally, most of these problems are NP-Hard [37], indicating that finding optimal schedules is likely to be computationally demanding and possibly prohibitive in practice.

Due to their formal computational difficulty, the majority of scheduling algorithms found in practice are heuristic approaches that focus on building feasible schedules quickly, sacrificing some degree of schedule optimality for low run-times. For example, [16] analyzes permutation-based optimization techniques, considering realistically sized Earth observing collection scheduling problems with power and thermal control constraints. Specific scheduling approaches considered included metaheuristics such as simulated annealing and genetic algorithms. Such metaheuristics were also applied to Earth observing collection scheduling problems in [36] and [52]. Other examples of heuristics and metaheuristics applied to collection scheduling problems are reported by [15] and [47]. Closely related optimization problems that focus on scheduling ground station communications are studied in [54] and [3], and a rolling-horizon strategy is combined with different heuristics for dynamic scheduling of Earth observing satellites in [9]. A number of different scheduling models and algorithms for earth observing satellites are reported in the references of [48].

An alternative to heuristics and metaheuristics for solving scheduling problems is the use of *exact* algorithms for solving mixed-integer programs. Mixed-integer (linear) programs, or MIPs, are a modeling formalism for optimization problems that is pervasive in the operations research and related optimization literature [5]. Exact algorithms for solving MIPs, such as branch-and-cut, are able to (1) obtain provably optimal solutions and (2) provide a bound on solution quality (i.e., the optimality gap) when the algorithm is terminated prior to global convergence [53]. In contrast, heuristics and metaheuristics provide no such guarantees or quality information. Over the last 10 to 15 years, commercial solvers for MIPs – including CPLEX (`https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer`) and Gurobi (`http://www.gurobi.com`) – have exhibited dramatic increases in performance, such that many industrial-scale problems previously considered insoluble by such approaches can now solve in minutes to tens of minutes of runtime. In this context, an ancillary component of our contribution in this paper includes the introduction of a simple, novel MIP model of the collection scheduling problem, in addition to demonstration that modern commercial MIP solvers can locate provably optimal or near-optimal solutions to these problems in tractable (i.e., operational) runtimes.

Independent of solution algorithm, the above approaches are implicitly *deterministic*, in that the values for parameters in the optimization problem are assumed to be known with certainty. However, in reality factors such as cloud cover, the movement of a phenomenon under observation (e.g, forest fires), and the availability of communication links for download are uncertain. A schedule optimized under deterministic parameter assumptions (e.g., observation quality for a given sensor at a particular time) may not be robust to perturbations in parameter values, and consequently may perform inefficiently or poorly in practice. Of course, the complexity of collection scheduling is further exacerbated when considering uncertainty, in terms of both modeling and solution approach. However, the benefits of directly addressing uncertainty may be significant, further im-

proving the utilization of expensive and over-subscribed remote sensing assets. The literature on models and techniques for proactively scheduling remote sensors against uncertainty stemming from cloud cover [48, 26], movement of a target under observation, system bandwidth resources, or collection utility is limited. In [26], the authors develop a stochastic mixed-integer program and make use of novel heuristics to maximize the rewards of completed collections and maximize the likelihood of successful collections. The model considered therein builds a feasible schedule for the FORMOSAT-2 satellite. The authors of [48] use a chance-constrained programming model and associated heuristics to proactively schedule against uncertain cloud-cover conditions. Our major contribution in this paper is the introduction of an optimization model for collection scheduling under uncertainty, and subsequent evaluation of the overall utility of this model in improving collection schedules when executed in practice. By utilizing efficient pre-processing, much of the complexity of scheduling sensors with dynamic orbits and fields-of-view and differing target or observation requirements can be encoded through careful definition of feasible collection window properties and indices.

1.2 Problem Definition

We now describe our collection scheduling models for remote sensing constellations, which are subsequently analyzed in [43]. We begin with a description of our high-level modeling assumptions in Section 1.2.1, and then introduce common notation and concepts in Section 1.2.2. Our primary deterministic coverage model is presented in Section 1.2.3, which generates collection schedules assuming predicted observation quality is identical to realized (i.e., actual) observation quality. We then present stochastic variants of this base coverage model in Section 1.2.6, which address potential deviations between predicted and realized observation quality. Finally, we present and discuss related deterministic and stochastic scheduling models developed by the LDRD team.

1.2.1 High-Level Modeling Assumptions

We assume that the performance of mobile sensors will be evaluated with respect to a fixed set of collection windows that are defined by:

- **Start time**: when the collection window starts.
- **Duration**: the length of time needed to completely sense a collection window. This is fixed and known before building a constellation schedule. Some collection windows may be divisible.
- **Physical location**: the location that needs to be observed; the precise requirements for observing a location will also depend on the nature of the target phenomenon.
- **Configuration**: the operational configuration required to observe a location; dependent on the target phenomenon to be observed.

- **Suitable sensors**: remote sensors with access to the physical location associated with the collection window and capability to collect the target phenomenon.
- **Quality**: the condition of the observation, which may be impacted by the physical location of the sensor relative to the observation, the time of day, and other factors. Quality is measured relative to the desired observation and there can be a minimum acceptable quality, below which the observation has no value to the user and is not considered to have been captured.
- **Priority**: the importance of this collection window relative to other collections.
- **Category**: a hierarchical system establishing the necessity of a collection window for a given sensor. Certain categories are required for proper sensor operation and therefore must be scheduled.

Managing sensors involves determining which collection window is to be scheduled on each sensor at each moment in time, and this should be done to optimize the value of the resultant set of observations. In general, we assume that time is suitably discretized. It is further assumed that the geometry of each sensor relative to the collection window is known accurately enough at each time-step so that collections can be scheduled on an appropriate sensor at a feasible time-step.

Finally, there are several general notes regarding our approach:

- Although we use the term *scheduling* here, there are no precedence constraints among most collection windows. Hence, a schedule of sensors simply reflects the quality, number, and selection of collection windows that they cover.
- There may be times where sensors cannot be employed (e.g., it is too dark to use an optical sensor). This could be represented using a low quality score for any collections on the sensor at those times or through an explicit modeling constraint. We employ both techniques.
- Mobile sensors may need regularly scheduled breaks where they are off-line (i.e., to refuel or recalibrate). This may be accomplished with constraints for the interval allowed between the collection windows or by assigning suitable priorities and time windows for these off-line breaks and treating them the same as collection windows. We have chosen the latter.
- Depending on a variety of criteria, multiple sensors may be able to cover a collection window simultaneously (though perhaps with different quality scores).

In practice, orbital propagators can be used to produce collection target and sensor relative geometries at a specified time interval, such as minutes or seconds, for the entirety of the scheduling horizon. This information can be used with customer supplied collection requests to generate lower- and upper-bounds on the collection window start and end times. Further constraints from a collection requirement, such as specific lighting conditions, specific sensor-target geometry constraints, or timeliness of information help further restrict the feasible sensor and time indices over which collection windows are defined. This information, combined with sensor and corresponding

configuration capability models, also serves to define the time-dependent quality values of collection windows. In general, these values need not be linear with time as geometric and physical parameters that determine the quality of complex collections are commonly nonlinear. Effective configuration quality models are defined to capture the relative quality of collections, according to geometric, lighting, configuration sensitivity, and timeliness of information considerations, across sensors and timesteps as well as relative to collection request requirements.

1.2.2 Common Definitions and Notation

Both our deterministic and stochastic optimization models are based on a simple mixed-integer programming "coverage" model for collection scheduling. Let $\mathscr{I} = \{1,\ldots,I\}$ denote the set of sensors that are being managed and let $\mathscr{T} = \{1,\ldots,T\}$ denotes the length of the scheduling horizon, i.e., the number of time steps that are to be scheduled. We assume each sensor shares the same scheduling horizon, such that $\mathscr{T}_i = \{1,\ldots,T\}\ \forall i \in \mathscr{I}$. This assumption can, however, be straightforwardly relaxed. For simplicity, we assume all sensors are distinct, even if they are co-located on a particular platform. In practice, any co-location information will be embodied in the collection window data, which implicitly captures all orbital dynamics.

Let $\mathscr{K} = \{1,\ldots,K\}$ denote the set of collection windows to be scheduled. For each collection window $k \in \mathscr{K}$, we define:

- (e_k, l_k) : the time window over which the collection window k can be *started*. By convention, $e_k, l_k \geq 1$, and we require $e_k \leq l_k$ and $e_k, l_k \leq T$.
- d_k : the required duration of the collection. We assume $d_k \leq T$.
- q_{ikt} : the (information) quality associated with assigning collection window k to sensor i at time t. By convention, $q_{ikt} \in [0,1]$.
- $q_k^{\min}$: the minimum acceptable quality required for collection window k, if scheduled. By convention, $q_k^{\min} > 0$.
- p_k : the priority associated with collection window k. By convention, $p_k \in [0,1]$.

We assume that the quality associated with assigning a specific sensor to a given collection window can be pre-computed in advance, e.g., via sensor simulation packages that may incorporate sensor-specific field-of-view geometry computations. Similarly, we assume e_k and l_k can be pre-computed, specifically using orbital dynamics packages.

In addition to the above attributes for each individual collection window $k \in \mathscr{K}$, we assume that the aggregate set of collection windows $\mathscr{K}$ is partitioned into three subsets or categories, which capture the necessity of particular collection windows being scheduled on specific sensor or sensors. Such categories are representative of the broad categories of collection windows found in practice. These categories are defined as follows:

- **Category 1** collection windows are unique to a given sensor, i.e., the collection window may only be assigned to one sensor. Category 1 collection windows must be scheduled, and therefore their priority p_k is generally set to 1.0. Examples of Category 1 collection windows include collections scheduled to address forced solar outages or other sensor safety restrictions.

- **Category 2** collection windows are also unique to a given sensor. Category 2 collection windows cannot overlap with Category 1 collection windows, for any given sensor. Category 2 collection windows generally have a high priority. Examples include collections associated with periodic calibration and/or maintenance activities.

- **Category 3** collection windows represent the largest proportion of collection windows. In general, Category 3 collection windows need not be restricted to a single sensor. Category 3 collection windows cannot overlap with Category 1 collection windows, for any given sensor. Category 3 collection windows generally have lower priorities than Category 2 collection windows. These can pre-empt Category 2 collection windows. Example of Category 3 collection windows include earth observation and other customer requested collections.

1.2.3 Deterministic Optimization Model Description

We now introduce our deterministic mixed-integer programming "coverage" model for collection scheduling. Let δ_{ikt} denote a binary decision variable where $\delta_{ikt} = 1$ if and only if sensor i is scheduled to execute collection window k starting at time period t. For convenience, we introduce auxiliary variables ω_k to denote whether or not collection window k has been scheduled at any time step, on any sensor. The ω_k are computed via the following constraint:

$$\omega_k = \sum_{i \in \mathscr{I}} \sum_{t \in \mathscr{T}} \delta_{ikt}, \forall k \in \mathscr{K}. \tag{1.1}$$

We observe that the ω_k variables are implicitly binary, due to the binary restrictions on the δ_{ikt}.

As discussed above in Section 1.2.2, all Category 1 collection windows must be scheduled, while Category 2 and 3 collection windows may be deferred due to limited sensor availability. Let $\mathscr{K}_1 \subseteq \mathscr{K}$ denote the set of indices for Category 1 collection windows. The following constraint then ensures that all Category 1 collection windows are scheduled:

$$\omega_k = 1, \forall k \in \mathscr{K}_1. \tag{1.2}$$

Next, let $\mathscr{C}(k, \bar{t})$ denote the set of time steps for which collection window k could have been started prior to time step $\bar{t}$ *and* for which scheduling k at those time steps would conflict with a distinct collection window scheduled to be executed at time step $\bar{t}$. Formally,

$$\mathscr{C}(k,\bar{t}) = \{\max(e_k,\bar{t}-d_k+1),\ldots,\min(l_k,\bar{t}-1)\}. \tag{1.3}$$

The purpose of the $\mathscr{C}(k,\bar{t})$ is to enumerate the set of potential sensor scheduling conflicts. These sets are used in the following constraint, to prevent contention for sensor resources:

$$\sum_{k\in\mathscr{K}} \delta_{ik\bar{t}} \leq 1 - \sum_{k\in\mathscr{K}} \sum_{t\in\mathscr{C}(k,\bar{t})} \delta_{ikt}, \forall \bar{t} \in \mathscr{T}, i \in \mathscr{I}. \tag{1.4}$$

In words, this constraint prohibits scheduling a collection window k at time $\bar{t}$ if another collection window was scheduled too recently, as defined by $\mathscr{C}(k,\bar{t})$. While the number of combinations of k and $\bar{t}$ is large, they can be quickly computed off-line.

The choice of optimization objective in collection scheduling is a matter of debate and, to some degree, preference. Sensor operators and planners have historically relied extensively on priority-based collection scheduling. Collection scheduling models can also capture the dependency of sensor performance on collection time and/or selected sensor, to enhance realism and improve overall utility of a constellation. Nearly universally, collection windows are weighted by a numeric priority such that trade-offs can be adjusted by sensor experts and/or customers.

Driven by our associated research, we choose to promote schedules in which long-duration, high-priority collection windows are scheduled to those sensor-time combinations that result in high-quality observations. Collections with longer durations are preferred in order to reduce wear-and-tear on limited life components, and can be de-emphasized through minor modification of the objective function introduced below. For convenience, let $\delta = \{\delta_{ikt}\}_{\forall i\in\mathscr{I},k\in\mathscr{K},t\in\mathscr{T}}$. We then define the optimization objective for the deterministic coverage model as

$$f(\delta) = \sum_{i\in\mathscr{I}} \sum_{k\in\mathscr{K}} \sum_{t\in\mathscr{T}} \frac{(\delta_{ikt})(p_k)(d_k)(q_{ikt})}{\alpha} \tag{1.5}$$

where the scalar α denotes a normalization constant, defined as follows. Let q_k^* denote the maximum potential quality collection window k can take over the scheduling horizon. We then define $\alpha = (\sum_{k\in\mathscr{K}} p_k d_k q_k^*)/100$ such that an objective function value of 100 will correspond to a sensor schedule in which all collection windows are scheduled to realize their highest potential quality.

Combining the family of constraints defined above with the objective function (1.5) yields the following deterministic coverage model for collection scheduling:

$$
\begin{array}{lll}
\max & f(\delta) & \\
\text{s.t.} & \omega_k = \sum_{i \in \mathscr{I}} \sum_{t \in \mathscr{T}} \delta_{ikt} & \forall k \in \mathscr{K}, \\
& \omega_k = 1 & \forall k \in \mathscr{K}_1, \\
& \sum_{k \in \mathscr{K}} \delta_{ik\bar{t}} \leq 1 - \sum_{k \in \mathscr{K}} \sum_{t \in \mathscr{C}(k,\bar{t})} \delta_{ik\bar{t}}, & \forall \bar{t} \in \mathscr{T}, i \in \mathscr{I}, \\
& \delta_{ikt} = 0 & \forall i \in \mathscr{I}, k \in \mathscr{K}, \text{and } t < e_k \text{ or } l_k < t, \\
& \omega_k \in \{0,1\} & \forall k \in \mathscr{K} \\
& \delta_{ikt} \in \{0,1\} & \forall i \in \mathscr{I}, k \in \mathscr{K}, t \in \mathscr{T}.
\end{array} \tag{1.6}
$$

The resulting coverage model is relatively simplistic, despite the complex nature of the underlying scheduling problem. We achieve this simplicity by both discretizing the scheduling horizon and shifting much of the computational burden to data processing associated with generation of the input parameters. Specifically, key collection window data – generated exogenously to the optimization model – specify all information related to time windows (a function of orbital dynamics) and observation quality for given sensors (a function of sensor models). Consequently, complex physics associated with these computations are left out of the explicit model formulation.

1.2.4 On Model Solvability

We have encoded the above deterministic model using the open-source Pyomo modeling language; see [19] and `http://www.pyomo.org` for additional details. We solve the resulting models on a 64-core workstation with 2.4GHz AMD processors and 512GB of RAM, using the commercial CPLEX mixed-integer programming solver (version 12.5). While solve times are not the primary focus of this paper, we do briefly report the following statistics, primarily to demonstrate the potential for use of exact methods in operational contexts.

Instances with a single sensor, $T = 1440$, and approximately 240 collection windows typically possess between 60K and 80K variables and 1600-1700 linear constraints, with between 800K and 1.2M non-zero coefficients in the constraint matrix. Instances with two sensors, $T = 1440$, and approximately 500 collection windows typically possess between 180K and 240K variables and 3300-3400 linear constraints, with between 2.4M and 3.2M non-zero coefficients in the constraint matrix. The one-sensor instances solve to within 1% of optimality in between 50 and 200 seconds, while two-sensor instances solve to within the same optimality gap within between 700 and 1400 seconds.

1.2.5 Enforcing Minimal Quality Levels

We have extended the deterministic optimization model introduced above to capture the notion of a minimum acceptable level of quality associated with a collection window. In particular, we observe that formulation (1.6) permits the scheduling of collection windows at times during which

quality is very low, as a means to fill the schedule while obtaining some increase in overall schedule performance. However, in many if not most real-world situations, such low-quality observations provide little or no information value, and are therefore not worth the associated collection costs (e.g., bandwidth and memory). To address this issue, we consider two approaches to capture this general notion via extensions to formulation (1.6).

In our first approach, we introduce a new binary variable for each collection window, to denote whether or not the collected information quality (if any) was super-threshold. A quality score below (above) the minimum acceptable quality threshold is instead replaced by a score of zero (one). We define $q_k^{\min}$ to be the minimum acceptable quality for a collection window k. The variable $q_{ikt}^{'}$ is used to determine whether or not the resulting quality was sub or super-threshold, and is determined as follows:

$$\begin{aligned} &\text{if } q_{ikt} < q_k^{\min}: \\ &\quad q_{ikt}^{'} = 0 \\ &\text{else:} \\ &q_{ikt}^{'} = 1, \forall i \in \mathscr{I}, k \in \mathscr{K}, t \in \mathscr{T}. \end{aligned} \tag{1.7}$$

In our second approach, we add a constraint to the model in order to simply threshold quality below a given value, such that if a collection window quality was less than $q_k^{\min}$ it is set to 0. This approach also discourages low quality collection windows from being scheduled, but differs from formulation (1.7) in that there can be a difference between quality scores that exceed the minimum acceptable qualities. This approach is formally expressed as follows:

$$\begin{aligned} &\text{if } q_{ikt} < q_k^{\min}: \\ &q_{ikt}^{'} = 0, \forall i \in \mathscr{I}, k \in \mathscr{K}, t \in \mathscr{T}. \end{aligned} \tag{1.8}$$

By enforcing a minimum quality, collection windows with large time windows can be scheduled according to time- and sensor-dependent quality. Subsequently, if a certain level of quality cannot be achieved, it represents an opportunity to prevent sensor resources from being wasted by producing observations that are unsatisfactory and instead scheduling an alternative collection window that meets its respective quality threshold. If one of the constraints above are used, the rest of the model is equivalent to (1.6), except that the newly generated $q_{ikt}^{'}$ is used instead of the previous q_{ikt}.

1.2.6 Stochastic Optimization Model Description

We now introduce variants of the coverage model for collection scheduling that explicitly account for uncertainty in how factors such as weather (e.g., cloud cover) impact observation quality. We

note that uncertainty about weather is a proxy for a broader set of factors that are difficult to predict, including target behaviors (e.g., movement of a forest fire). The goal of stochastic coverage models is to create schedules that are resilient to adverse performance impacts that may be caused by uncertain predictions regarding future conditions. We begin in Section 1.2.7 with an overview of stochastic optimization models, specifically stochastic mixed-integer programs. We then introduce our first stochastic collection scheduling optimization model in Section 1.2.8, in which weather for the next day is unknown, but a number of potential realizations of weather are available. Our second stochastic model is then introduced in Section 1.2.9, which allows for the modeling of both weather forecast updates during next-day operations, and re-scheduling to account for these updates.

1.2.7 Stochastic Mixed-Integer Programming: An Overview

We now briefly describe the general formulation of a stochastic mixed-integer programming (SMIP) problem. For a more detailed introduction to this topic, we refer the reader to [4] and [39]. Our primary stochastic collection scheduling model consists of two discrete decision stages; subsequently, we will consider three discrete decision stages in Section 1.2.9. The first decision stage aims to schedule all Category 1, 2, and 3 collection windows. Then, in contrast to our deterministic collection scheduling model, uncertainty associated with cloud cover is realized (due to the passage of time). Such uncertainty resolution conceptually occurs between the first and second decision stages. The "decisions" in the second stage then consist of computing the quality of all assigned observations, given realized cloud cover. The first stage decision must be made here-and-now, without full information concerning future events. The second stage decisions (in this case, computations) are made after all information is realized. Second stage decisions are commonly referred to as *recourse* decisions. Uncertainty in stochastic mixed-integer programming is most often expressed in the form of a *scenario tree*, which is a collection of potential future realizations of uncertain quantities (e.g., cloud cover) with associated probabilities.

We formalize this general decision-making process. Let x denote the first stage decision variable vector and let $y(\xi)$ denote the second stage recourse decision vector for scenario ξ. A generic two-stage SMIP can then be specified as

$$\begin{aligned} &\max\ f(x) + \mathbb{E}[g(x, \tilde{\xi})] \\ &\text{s.t. } Ax \leq b \\ &\qquad x \in X, \end{aligned} \tag{1.9}$$

where the set $X \subseteq \mathbb{R}^{n_1}$ abstractly captures discrete (e.g., binary or integer) restrictions on some or all components of x, $f(x)$ denotes the first stage objective function, $\mathbb{E}$ denotes the mathematical expectation operator and $\mathbb{E}[g(x, \tilde{\xi})]$ denotes the expected value of second stage costs. The vector $\tilde{\xi}$ denotes a multivariate random variable defined on a probability space $(\Xi, \mathscr{F}, \mathscr{P})$ and abstractly represents parametric uncertainty in the SMIP. This probability space is in practice typically approximated by a discrete set of realizations Ξ, i.e., scenarios. Individual scenarios in Ξ are denoted

by ξ, and possess a corresponding probability of p_ξ. Necessarily, $\sum_{\xi\in\Xi} p_\xi = 1$. The objective in formulation (1.9) can then be expressed as:

$$\max\ f(x) + \sum_{\xi\in\Xi} p_\xi g(x,\xi) \tag{1.10}$$

For a specific realization $\xi \in \Xi$ of $\tilde{\xi}$, the second stage or recourse problem $g(x,\xi)$ can be specified as follows:

$$\begin{aligned} g(x,\xi) = \max\ & h(\xi)^\top y(\xi) \\ \text{s.t. } & W(\xi)y(\xi) \le r(\xi) - T(\xi)x \\ & y(\xi) \in Y. \end{aligned} \tag{1.11}$$

In model (1.11), $Y \subseteq \mathbb{R}^{n_2}$ abstractly captures discrete restrictions on some or all components of $y(\xi)$, $h(\xi)$ denotes coefficients in the second stage objective function, $W(\xi) \in \mathbb{R}^{m_2\times n_2}$ and $T(\xi) \in \mathbb{R}^{m_2\times n_1}$ denote constraint coefficient matrices (respectively independent of and dependent on first stage decisions), and $r(\xi) \in \mathbb{R}^{m_2}$ denotes a right-hand side vector on the constraint matrix. A scenario ξ defines one potential realization of the stochastic problem data $\{h(\xi), W(\xi), T(\xi), r(\xi)\}$. If $W(\xi) = W$ for all $\xi \in \Xi$, the SMIP is said to have *fixed recourse*. Otherwise, the SMIP is said to have *random recourse*.

In SMIPs, the structural location of randomness can greatly impact on the difficulty of solving the problem [31]. Such difficulties have driven the design of decomposition algorithms to more efficiently solve specific classes of problem, e.g., see [32]. In general, having *random recourse* makes the problem harder to solve due to the fact the the constraint matrix changes with respect to ξ and thus scenarios may not share the same solutions. Problems having relatively fewer random parameters are computationally desirable.

1.2.8 A Two Stage Model

In our two-stage stochastic collection scheduling model, the first stage represents all variables and constraints associated with determining a schedule, i.e., formulation (1.6) without the objective function. In contrast, the second stage represents those modeling components required to compute schedule performance once a specific weather scenario is realized – given a fixed first-stage schedule. The overall optimization objective is to maximize the probability-weighted sum of schedule performance across all weather scenarios. In Figure 1.1, we provide an illustrative example of a two-stage scenario tree with 3 scenarios. In this example, the sole source of uncertainty is the time during which a weather system obscures a particular geographic region of interest.

To compute the realized quality associated with a set of scheduled collection windows, we first introduce a subset $\mathscr{K}_v \subseteq \mathscr{K}$ to represent those collection windows whose quality can be impacted

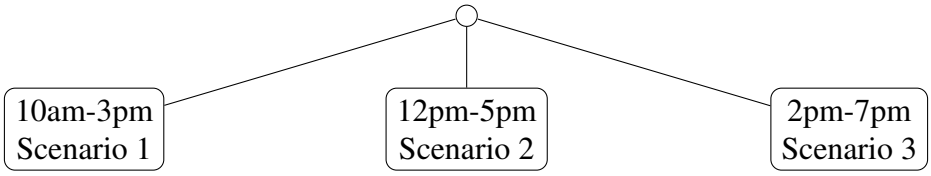


Figure 1.1. An illustrative scenario tree for a two-stage stochastic collection scheduling model. The example consists of three scenarios, varying in the arrival time of a weather front that brings cloud cover. The top "row" of the model represents the first decision stage (i.e., scheduling), while the bottom "row" of the model represents the second decision stage (i.e., schedule evaluation). The times represent when the cloud front will arrive and thus set $\mathscr{K}_V$ collection windows to have a quality measure of 0.

by cloud cover. We assume that all collection windows not in $\mathscr{K}_v$ receive a quality score that is independent of cloud cover. Then, for each time period $t \in \mathscr{T}$ and scenario $\xi \in \Xi$, we let $c_{t\xi}$ denote the percentage of cloud cover over the geographic region of interest; clearly, $c_{t\xi} \in [0,1]$. For simplicity, we ignore any potential spatial component of $c_{t\xi}$ – all collection windows $\mathscr{K}_v$ are equally impacted, independent of actual geographic proximity. However, this assumption can be easily relaxed, subject to data availability.

Next, we introduce variables and constraints to the second stage models (one per scenario) to account for cloud cover and its impact on realized quality. We assume that for all scheduled collection windows in $\mathscr{K}_v$, quality is an inverse linear function of cloud cover $c_{t\xi}$. The realized quality $q_{ikt\xi}$ in the second stage of each scenario is then computed as:

$$q_{ikt\xi} = q_{ikt}(1 - c_{t\xi}) \forall i \in \mathscr{I}, k \in \mathscr{K}_v, t \in \mathscr{T}, \xi \in \Xi. \tag{1.12}$$

For collection windows not impacted by clouds, we have realized quality:

$$q_{ikt\xi} = q_{ikt} \forall i \in \mathscr{I}, k \in \mathscr{K} - \mathscr{K}_v, t \in \mathscr{T}, \xi \in \Xi. \tag{1.13}$$

The optimization objective for a single scenario in our two-stage stochastic collection scheduling model is structurally identical to that of (1.5), with the exception that the referenced $q_{ikt\xi}$ reside in the second stage of scenario models.

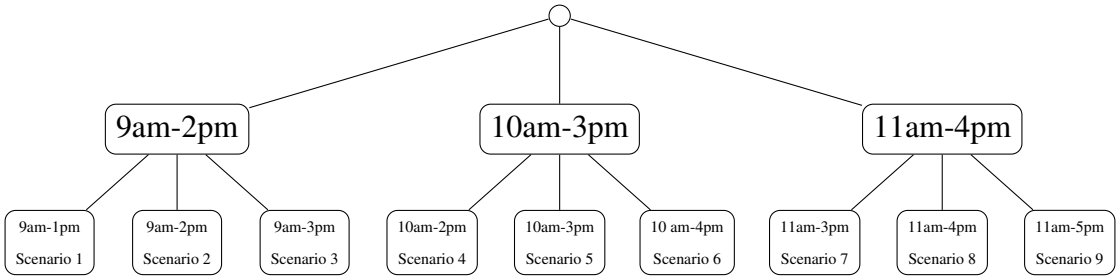


Figure 1.2. An illustrative scenario tree for the three stage stochastic collection scheduling model. The top, middle, and bottom rows denote the first, second, and third stages in the model, respectively. Given the realization of the second stage, you are left with three new scenarios in the third stage. The times represent when the cloud front will be present and thus set $\mathscr{K}_V$ collection windows to have a quality measure of 0.

1.2.9 A Three Stage Model

Our two-stage model lacks a key feature found in advanced stochastic programming models – logical recourse, or the ability to adapt or respond to changing conditions. Computing realized quality quantities is structural recourse, in that there are variables that are computed in the second stage, following resolution of cloud cover uncertainty. However, these variables do not represent operational decisions, such as those involved in re-scheduling.

To address this shortcoming, we now introduce a three stage stochastic collection scheduling model. In our three stage model, the first stage models the planned schedule for the entire scheduling horizon, before any specific information regarding realized weather patterns is available. The second stage represents schedule adjustments once the front arrival times are known / realized. Finally, the third stage consists of schedule adjustments following realization of the front departure time.

Before formalizing the variables and constraints associated with the second and third stages, we first partition the time window $\mathscr{T}$ into two subsets $\mathscr{T}2$ and $\mathscr{T}3$, such that $\mathscr{T} = \mathscr{T}2 \cup \mathscr{T}3$. The subset $\mathscr{T}2$ contains time steps for which the schedule is executed prior to receiving new weather predictions. Analogously, $\mathscr{T}3$ contains time steps for which the schedule is executed after new weather predictions are received.

Let $q2_{ikt\xi}$ be the new quality measure in the second stage based on the second stage cloud cover percentages, $c2_{t\xi}$. Let $\mathscr{K}_v$ represent all collection windows affected by clouds. Then $q2_{ikt\xi}$ can be computed as

$$q2_{ikt\xi} = q_{ikt}(1 - c2_{t\xi}), \forall i \in \mathscr{I}, k \in \mathscr{K}_v, t \in \mathscr{T}2, \xi \in \Xi. \tag{1.14}$$

The third stage cloud cover percentages are given as $c3_{t\xi}$ and indexed by all t in $\mathscr{T}3$ and all ξ in Ξ, so there are new cloud cover percentages for all times after new weather predictions are received. Let $q3_{ikt\xi}$ be the new quality measure based on cloud cover data given by $c3_{t\xi}$. Then $q3_{ikt\xi}$ can be computed as,

$$q3_{ikt\xi} = q_{ikt}(1 - c3_{t\xi}), \forall i \in \mathscr{I}, k \in \mathscr{K}_v, t \in \mathscr{T}3, \xi \in \Xi. \tag{1.15}$$

Similar to the deterministic model, two variables are created to denote the collection windows that were scheduled in stage one, ω_k and δ_{ikt}. δ_{ikt} is a binary variable that is set to 1 if collection window k is scheduled at time t on sensor i, and 0 if it is not scheduled. The variable ω_k denotes if collection window k has been scheduled at all, it is set to 1 if k has been scheduled and 0 if k has not been scheduled.

Two variables are also created to account for what has been scheduled before new weather information (in stage 2), called $\omega 2_{k\xi}$ and $\delta 2_{ikt\xi}$. Note that both $\omega 2$ and $\delta 2$ are indexed by scenario, because the realized weather scenario affects whether or not observations were successful. These function in the same way as ω_k and δ_{ikt} but they represent what was actually observed in the time window $\mathscr{T}2$.

Similarly, two variables are created to schedule collection windows after new weather information has arrived (stage 3), called $\omega 3_{k\xi}$ and $\delta 3_{ikt\xi}$. Again these variables function like ω_k and δ_{ikt} but instead represent what was actually observed in the time window $\mathscr{T}3$ and are indexed by scenario.

Let $\mathscr{C}(k,\bar{t})$ be the set of time steps where k could have been scheduled prior to time step $\bar{t}$ that would conflict with a new sensor starting an observation at time step $\bar{t}$. Thus

$$\mathscr{C}(k,\bar{t}) = \max(e_k, \bar{t} - d_k + 1), \ldots, \min(l_k, \bar{t} - 1). \tag{1.16}$$

The following equation (equivalent to equation (1.4) in the deterministic model) is used to ensure that multiple collection windows cannot be scheduled at the same time on the same sensor for stages 1 and 2,

$$\sum_{k \in \mathscr{K}} \delta_{ik\bar{t}} \leq 1 - \sum_{k \in \mathscr{K}} \sum_{t \in \mathscr{C}(k,\bar{t})} \delta_{ikt}, \forall \bar{t} \in \mathscr{T}, i \in \mathscr{I}. \tag{1.17}$$

The following equation mirrors (1.17) to ensure that multiple collection windows cannot be scheduled at the same time on the same sensor for stage 3.

$$\sum_{k \in \mathscr{K}} \delta 3_{ik\bar{t}\xi} \leq 1 - \sum_{k \in \mathscr{K}} \sum_{t \in \mathscr{C}(k,\bar{t})} \delta 3_{ikt\xi}, \forall \bar{t} \in \mathscr{T}, i \in \mathscr{I}, \xi \in \Xi. \tag{1.18}$$

To ensure that stage three can only schedule collection windows during the time $\mathscr{T}3$ the following constraint is used,

$$\delta 3_{ikt\xi} = 0, \forall i \in \mathscr{I}, k \in \mathscr{K}, t \in \mathscr{T}2, \xi \in \Xi. \tag{1.19}$$

The variable $\delta 2_{ikt\xi}$ is almost equivalent to δ_{ikt} over $\mathscr{T}2$, except if $q2_{ikt\xi}$ was 0 then δ_{ikt} registers that the collection window has been scheduled, although it has not actually been observed. To enforce this the following constraint is used,

$$\begin{aligned} &\text{if } q2_{ikt\xi} = 0: \\ &\quad \delta 2_{ikt\xi} = 0 \\ &\text{else:} \\ &\delta 2_{ikt\xi} = \delta_{ikt} \forall i \in \mathscr{I}, k \in \mathscr{K}, t \in \mathscr{T}2, \xi \in \Xi. \end{aligned} \tag{1.20}$$

Then, all ω_k variables are defined so that ω_k is 1 if the collection window k is scheduled and 0 if it is not.

$$\omega_k = \sum_{i \in \mathscr{I}} \sum_{t \in \mathscr{T}} \delta_{ikt}, \forall k \in \mathscr{K}. \tag{1.21}$$

$$\omega 2_{k\xi} = \sum_{i \in \mathscr{I}} \sum_{t \in \mathscr{T}2} \delta 2_{ikt\xi}, \forall k \in \mathscr{K}, \xi \in \Xi. \tag{1.22}$$

$$\omega 3_{k\xi} = \sum_{i \in \mathscr{I}} \sum_{t \in \mathscr{T}3} \delta 3_{ikt\xi}, \forall k \in \mathscr{K}, \xi \in \Xi. \tag{1.23}$$

From this point, whether or not a collection window is actually observed is expressed by the sum of $\omega 2_{k\xi}$ and $\omega 3_{k\xi}$. The following ensures that collection windows that are Category 1 will be observed either in stage 2 or stage 3,

$$\omega 2_{k\xi} + \omega 3_{k\xi} = 1, \forall k \in \mathscr{K}_1, \xi \in \Xi. \tag{1.24}$$

For all collection windows, this sum also must not be greater than one, so each collection window can be observed at most once:

$$\omega 2_{k\xi} + \omega 3_{k\xi} \leq 1, \forall k \in \mathscr{K}, \xi \in \Xi. \tag{1.25}$$

The objective function is slightly changed to incorporate the new stages as well as the new variables, as such there are three variables that quantify the value of each stage, $\mathscr{V}_1$, $\mathscr{V}_{2\xi}$, and $\mathscr{V}_{3\xi}$. These are set as:

$$\mathscr{V}_1 = \sum_{i \in \mathscr{I}} \sum_{k \in \mathscr{K}} \sum_{t \in \mathscr{T}} \frac{(\delta_{ikt})(p_k)(d_k)(q_{ikt})}{\alpha}. \tag{1.26}$$

$$\mathscr{V}_{2\xi} = \sum_{i \in \mathscr{I}} \sum_{k \in \mathscr{K}} \sum_{t \in \mathscr{T}} \frac{(\delta 2_{ikt\xi})(p_k)(d_k)(q2_{ikt\xi})}{\alpha}, \forall \xi \in \Xi. \tag{1.27}$$

$$\mathscr{V}_{3\xi} = \sum_{i \in \mathscr{I}} \sum_{k \in \mathscr{K}} \sum_{t \in \mathscr{T}} \frac{(\delta 3_{ikt\xi})(p_k)(d_k)(q3_{ikt\xi})}{\alpha}, \forall \xi \in \Xi. \tag{1.28}$$

The final objective function is the sum of $\mathscr{V}_{2\xi}$, the value of what was actually observed in the second stage and $\mathscr{V}_{3\xi}$, the value of what was actually observed in the third stage multiplied by the probability of scenario ξ being realized summed over all ξ in Ξ. This is expressed as follows,

$$f(\delta) = \sum_{\xi \in \Xi} p_\xi (\mathscr{V}_2 + \mathscr{V}_3) \tag{1.29}$$

1.2.10 On Model Solvability

We encoded the above stochastic models using the PySP sub-package of the Pyomo modeling language; see [49] for additional details. We solve the resulting models using the CPLEX MIP solver on the same 64-core workstation as used for solving the deterministic models.

Due to the number of scenarios used, our two-stage stochastic collection scheduling models are significantly larger than their deterministic counterparts. 100-scenario instances with a single sensor, $T = 1440$, and approximately 240 collection windows typically possess between 6 and 8 million variables, an equal number of constraints, and 90 to 120 million non-zero coefficients in the constraint matrix. 100-scenario instances with two sensors, $T = 1440$, and approximately 500 collection windows typically posses between 19 and 24 million variables, an equal number of constraints, and between 270 and 350 million non-zero coefficients in the constraint matrix. Yet, despite the size of these models, CPLEX is able to obtain solutions within 1% of optimality in between 600 and 1200 seconds for the one-sensor instances, and between 2000 and 3000 seconds for the two-sensor instances. A three-stage stochastic model with 2 sensors, $T = 1440$, 430 collection windows, and 9 scenarios possesses approximately 5.2 million variables and constraints and has 42.2 million non-zero coefficients in the constraint matrix. This model can be solved to an optimality gap of 0.23% in 2400 seconds.

While these run-times are slightly larger than those required in operational settings, they can be significantly reduced using decomposition strategies for solving stochastic programs, e.g., progressive hedging [38, 50].

1.2.11 Additional Models and Algorithms

Experiments using the generated benchmark data and the presented deterministic and stochastic models focused on applications to Earth observing satellites in geostationary orbit. However, the models were designed for the broad class of remote sensors including ground and space-based telescopes for space situational awareness (SSA), UAVs, unattended ground sensors, etc., of arbitrary orbital or motion characteristics. Members of the Texas A&M University team developed heuristics for the deterministic model that produce near-optimal schedules within seconds on desktop machines in [41]. An objective function penalizing schedule gaps is presented in [45]. We describe an extension to our deterministic model to allow for concurrent collections on a single sensor and present a stochastic model for proactively scheduling against the uncertainty of *ad hoc* collections in [44] and corresponding poster.

Additional experiments were conducted over the course of the project. Although we do not report on the results here, a class of constraints were added to the model in Equation (1.6) to allow certain collection windows to be divided and scheduled separately. The goal of this experiment was to assess the impact on schedule information gain if certain collections are allowed to be shortened. As expected, overall objective function value increased but runtime also increased as additional *a priori* collection windows were evaluated for scheduling.

Three additional modeling extensions were explored. First, a constraint to require certain collection windows to be scheduled concurrently on two distinct sensors was developed. If we modify the model in 1.6 with the following relaxations and constraints

$$\begin{array}{ll}
\omega_k = 1 & \forall k \in \mathscr{K}_1 \setminus \mathscr{K}_{cc}, \\
\omega_k \geq 1 & \forall k \in \mathscr{K}_1 \cap \mathscr{K}_{cc}, \\
\omega_k \leq 1 & \forall k \in \mathscr{K} \setminus \mathscr{K}_1 \cup \mathscr{K}_{cc}, \\
\sum_{t \in \mathscr{T}} \delta_{ikt} \leq 1 & \forall k \in \mathscr{K}_{cc}, \forall i \in \mathscr{I} \\
\sum_{m \in \mathscr{I}} (\sum_{t \in \mathscr{T}} \delta_{ikt} \cdot t - \sum_{t \in \mathscr{T}} \delta_{ikm} \cdot t) = 0 & \forall k \in \mathscr{K}_{cc}, \forall i, m \in \mathscr{I}, m \neq i \\
\omega_k \in \mathbb{N} &
\end{array} \tag{1.30}$$

where $\mathscr{K}_{cc}$ are the set of collection windows that must be scheduled concurrently on two separate sensors. The above constraints only work for models with exactly two sensors.

A model to exploit the structure of sensor scheduling problems where Category 2 collections are required to be scheduled was also explored. By aggregating the time-steps between Category 2 collections into "knapsacks", the size of the mixed-integer linear program can be significantly reduced with limited impact to the fidelity of the scheduling model. We modify the decision variable δ to instead be indexed by sensor ($i \in \mathscr{I}$), collection window ($a \in \mathscr{A}$), and knapsack ($k \in \mathscr{K}$) leading to the following mixed-integer linear program:

$$
\begin{array}{lll}
\max & \sum_{i\in\mathscr{I}}\sum_{a\in\mathscr{A}}\sum_{k\in\mathscr{K}}\frac{(\delta_{iak})(p_a)(d_a)(q_{iak})}{\alpha} & \\
\text{s.t.} & W_{min}\le U_{ik}\le W_{max} & \forall i\in\mathscr{I},\forall k\in\mathscr{K}, \\
& \sum_{k\in\mathscr{K}}U_{ik}\le T & \forall i\in\mathscr{I}, \\
& \sum_{a\in\mathscr{A}}\delta_{iak}\cdot d_k\le U_{ik} & \forall i\in\mathscr{I}, \\
& \omega_k\le 1 & \forall k\in\mathscr{K}\setminus\mathscr{K}_1\cup\mathscr{K}_{cc}, \\
& \omega_k\in\{0,1\} & \\
& U_{ik},W_{max},W_{min}\in\mathbb{N}, &
\end{array}
\tag{1.31}
$$

where W_{min} and W_{max} represent the minimal and maximal duration allowed between Category 2 collections and U_{ik} represents the duration of sensor i's k-th knapsack. As before, category one collection windows must be scheduled and d_a, p_a, and q_{iak} represent collection window a's duration, priority, and quality when scheduled on sensor i's k-th knapsack, respectively.

Finally, we are investigating, in collaboration with Texas A&M, a model extension that would expand δ_{ikt} of Equation (1.6) to include an index for sensor configuration. Collection windows under this model can be scheduled according to different potential sensor configurations and allowing multiple collections to run concurrently on a sensor as long as the collections can be scheduled with the same configuration. The goal of this model is to sacrifice some model simplicity for the ability to simplify location-based bonusing optimization and push it into the pre-processing stages of the sensor scheduling problem. For example, if a sensor boresite can cover multiple collection window targets when centered over a single location and with a specific sensor operation configuration, note this in the feasible indices of δ. While boresite locations outside of centered target locations will not be considered, bonusing will be effectively passed to the mixed-integer linear model for optimal scheduling. The model is described below:

$$
\begin{array}{lll}
\max & \sum_{i\in\mathscr{I}}\sum_{k\in\mathscr{K}}\sum_{t\in\mathscr{T}}\sum_{m\in\mathscr{M}}\frac{(\delta_{iktm})(p_k)(d_k)(q_{iktm})}{\alpha} & \\
\text{s.t.} & \omega_k=\sum_{i\in\mathscr{I}}\sum_{t\in\mathscr{T}}\sum_{m\in\mathscr{M}}\delta_{ikt} & \forall k\in\mathscr{K}, \\
& \omega_k=1 & \forall k\in\mathscr{K}_1, \\
& \Delta_{i\bar{t}m}=\sum_{k\in\mathscr{K}}\sum_{t\in\mathscr{C}(k,\bar{t})}\delta_{ik\bar{t}m}, & \forall\bar{t}\in\mathscr{T},i\in\mathscr{I},m\in\mathscr{M}, \\
& \Omega_{itm}\cdot|K|\ge\Delta_{itc} & \forall\bar{t}\in\mathscr{T},i\in\mathscr{I},m\in\mathscr{M}, \\
& \Omega_{itm}\le\Delta_{itm} & \forall\bar{t}\in\mathscr{T},i\in\mathscr{I},m\in\mathscr{M}, \\
& \sum_{c\in\mathscr{C}}\Omega_{itm}\le 1 & \forall\bar{t}\in\mathscr{T},i\in\mathscr{I}, \\
& \omega_k\in\{0,1\} & \forall k\in\mathscr{K} \\
& \delta_{iktm}\in\{0,1\} & \forall i\in\mathscr{I},k\in\mathscr{K},t\in\mathscr{T}, \\
& \Delta_{itm}\in\mathbb{N} & \forall i\in\mathscr{I},t\in\mathscr{T},m\in\mathscr{M}, \\
& \Omega_{itm}\in\{0,1\} & \forall i\in\mathscr{I},t\in\mathscr{T},m\in\mathscr{M}.
\end{array}
\tag{1.32}
$$

In the above model, Δ_{itm} is a convenience variable representing the number of collection windows that have started or will start on sensor i at time t with configuration m. An indicator variable Ω_{itm}

is used to indicate whether or not at least on collection with configuration m is running or will start on sensor i at time t.

Chapter 2

Footprint Placement Optimization

2.1 Problem Definition and Background

We consider the *footprint problem* of robustly selecting a minimal set of sensor images (footprints) whose union covers a Region Of Interest (ROI) on the earth. This is also known as the *mosaic* problem. Mosaics are common to aerial and space-based imaging and are collected with the primary goal of providing imagery of a much larger area of the Earth than is possible with single images. Ensuring that there are no gaps between constituent images is important to the effective utilization of the mosaic. In some settings, significant image overlap is desired. Efficient placement of constituent images can impact both mosaic quality and the sensor time needed to produce a useful mosaic.

We represent the ROI and the sensor footprints as polygons (with holes) or circles. Since the combinatorial complexity of the ROI and footprint are not key features in our problem, we rarely discuss these and the reader may consider the footprint to be approximately a circle or square. However, our decision was to not specialize our algorithms for circles or squares, but instead focus on algorithms that would work well in practice on the broader class of nearly-convex footprint shapes with good aspect ratios.

This problem lies in the family of geometric coverage problems. Many approaches to geometric coverage exploit the structure of the shape used to cover the domain. For example, circles lead to some elegant equations that make the problem amenable to gradient descent optimization [40]. There are also specialized solutions for axis-aligned rectangles. However, it is typically the case for general shapes to be theoretically hard to find optimal solutions efficiently. Hence heuristics for approximate solutions are common. While squares and circles are common examples of camera images, circular and square pictures of the earth from orbit lead to covering sections of the earth that are not circular or square [55]; see Section 2.7.3. That is, the projection of a circle onto a sphere is not, in general, a circle. Alternatively, one may retain circular or square images by projecting the ROI onto the camera image plane. This makes the description of the ROI more complicated, and there are fidelity issues for grazing camera angles, and it was unclear how this approach would work with time-varying sensor positions and interact with the scheduling problem.

One category of practical covering algorithms is the "shifting technique" [18, 17]. One starts with a set of footprints that is feasible, meaning it covers the ROI, then makes local changes

to try to reduce the size of the set. It is easy to get an initial feasible solution, for example a "greedy cover" where one iteratively adds a footprint covering the most possible area of the ROI that remains uncovered, or a GreedyKCenter where one places footprints as far as possible from prior footprints, or a regular packing. One local improvement technique would be to throw away k connected regions, and try to insert $k-1$ regions instead while still covering the entire ROI. For algorithmic efficiency, k must be small, say 2 or 3. We know this works for disks [29] but is challenging for arbitrary shapes [18]. The replace-with-fewer-disks suggestion is similar to "Sifted Disks" [11] and "Disk Tuning" [13]. Other variations not only reduce the number of disks while maintaining coverage, but can also maintain or improve other geometric properties of the set [1, 10].

Another covering-algorithm category includes relaxation techniques, where one reformulates problem P into Q where it is possible to find an optimal solution to Q, and this solution is also good (but sub-optimal) for P. Our approach falls into this category. We redefine the problem as a discrete one, amenable to mixed-integer linear Programming (MIP). Sariel Har-Peled [18] suggests an approach very similar to ours, turning a cover-a-polygon-by-disks problem into a discrete optimization problem by first point-sampling the polygon without large gaps, then covering the discrete set by disks, and expanding their radii to ensure polygon coverage. These reformulated problems are geometric versions of the classical "min set cover" problem.

Merritt [28] considers a very similar problem, motivated by radar coverage of the U.S. The solution is an adaptation of the algorithm of Kazazakis and Argyros [24], based on recursive subdivision of the ROI. She also considers the problem of limited resources, covering only up to some fraction of the ROI. If a polygon (subregion of the ROI) is sufficiently covered by one footprint at its center, it is considered covered and discarded. Otherwise, the polygon is subdivided. Merrit provides experimental performance data.

Daniels [7] considers the translational motion of polygonal covering shapes to cover a set of polygonal target items. As with many other coverage problems, this variation is an NP-hard problem. As in our problem, the covering (footprint) shapes are allowed to overlap each other as well as extend beyond the bounds of the target items (ROI). The solution method partitions the target items into triangles. The reformulation is a combinatorial optimization problem, to maximize the number of triangles covered. The solution method iteratively uses a Lagrangian heuristic. If the Lagrangian heuristic did not cover all the triangles, an uncovered triangle is subdivided and the heuristic is rerun. Karen Daniels kindly shared her source code with us.

Initially, we thought there might be some useful background in the "art-gallery" algorithm literature [8]. The art-gallery problems concern ensuring that a polygon (the gallery) is covered by line-of-sight cones with apex within the polygon (the guards). There are variations for time-varying coverage, ensuring that there is no uncovered path from one side of the polygon to the other, and variations for line-guards and moving or rotating guards. However, each of these algorithms appear to be specialized based on details of the problem definition, and are not easily extensible to covering by shapes such as limited-radius circles or rectangles.

2.2 Converting "Cover a ROI with Footprints" into a Discrete Optimization Problem

2.2.1 Algorithm Summary

The footprint problem P of finding the minimum number of *footprints* and their placements that can cover a Region Of Interest (ROI) can be converted into a discrete optimization problem Q that approximates P. A footprint placement is modeled as a translation of the footprint shape, which is assumed to be a fat and nearly-convex polygon. The ROI is modeled as a planar polygon, with perhaps some curvature and isolated lines and points. The idea is to create a set of discrete coverage points within the ROI, and a set of nearby discrete placement points of where to place footprints. We then solve an optimization problem of where to place the footprints such that all of the coverage points are far enough inside at least one footprint, so that all of the ROI is inside the union of footprints.

Coupling with Scheduling. One value of describing the footprint and sub-footprint placement problems as MIP optimization problems, as opposed to more geometric algorithms, is that they can be coupled directly into scheduling optimization problems. We have not implemented this coupling.

The procedure begins with the user choosing a geometric approximation parameter ε. The smaller the ε, the higher the quality of the solution and the longer the running time. The ROI is covered by some set of ε-radius circles (not the footprint), whose centers are *coverage points*. Hence the distance from any point of the ROI to the nearest coverage point is at most ε. We generate a second set of *placement points*; again, using more points improves solution quality at the cost of additional runtime. We consider placing a footprint at each placement point, and compute the set of coverage points within the footprint by at least ε distance. (Conceptually, one may consider a footprint shrunk by distance ε.) For each placement point, we have a Boolean 0–1 decision variable to decide whether to place a footprint there. The optimization problem Q is to minimize the sum of 0–1 decision variables (number of footprints). For each coverage point, we have a constraint that it is covered at least once, modeled by constraining that the sum of decision variables that actually cover it is at least one. Any solution to Q is a solution to P, meaning the ROI is covered by footprints, but the optimal solution to Q may be suboptimal for P, in that it may have been possible to cover the ROI with fewer footprints.

2.2.2 Sampling for Coverage Points

We generate the covering circles through a variant of "Simple MPS" [12] , which places random points at least r apart until every ROI point is within r of a sample, where $r \leq \varepsilon$. Alternatively one may use some other Maximal Poisson-Disk (MPS) sampling, or a Delaunay refinement algorithm, with maximum Voronoi vertex to sample point distance ε. One could even use a structured lattice pattern. The radius does not affect the correctness (whether the footprints cover the ROI) but

does affect the quality (number of footprints) and efficiency (time to solve). Simple MPS divides the ROI into squares, and samples points uniformly from the squares. Accepted samples are the centers of disks of radius ε. A square is discarded if it is covered by a single disk. A sample is rejected if the point lies outside the ROI or inside a prior (previously-accepted) disk. After some constant number of sample attempts, such as 3 times the number of squares, all the squares are divided into 4. Squares are discarded if they lie outside the ROI or inside a disk. Repeat until the remaining squares are the size of machine precision. In practice Simple MPS is linear in time and memory, and is easy to code because it only relies on simple and reliable geometric primitives such as is-point-in-circle and is-point-in-square.

2.2.3 Sampling for Placement Points

The main difference between this step and sampling for coverage points, is that placement points may be outside the ROI. We still restrict placement points to the convex hull of the ROI. The optimal placement may use points outside the ROI if the ROI is non-convex. For example, if the ROI is an annulus one might want to place a footprint near the annulus center.

Here we introduce the concept of an *anchor* point for the footprint that describes where the footprint lies if we place its anchor point on a placement point. The *footprint radius* is the farthest distance from a footprint point to its anchor point. A natural choice for the anchor is the centroid of the footprint. Because of the consequences of the footprint radius, and how far outside the ROI one wants to explore placing a footprint, there is no perfect choice except for circles. To minimize the radius, one would use the center of the smallest enclosing circle of the footprint. In our implementation, we use the centroid.

So, we rerun Simple MPS, but consider the domain to be the region inside both the maximum footprint-radius of the ROI, and the convex hull of the ROI. To determine if a sample point is inside this expanded domain, we use the same kind of distance calculations as described in the next subsection. For simplicity we initialize the set of placement points to be the a subset of coverage points, then fill in the rest of the domain, rather than generating them independently from scratch. However, we use a larger sampling radius, e.g. $\sqrt{2}\varepsilon$, in order to keep the MIP Q small enough to be solved quickly.

2.2.4 Covering Samples by Placements

Polygonal (straight sides) ROI and footprints are not needed. It is straightforward to convert a curved footprint into a polygonal one inside it. Conversely, for a curved ROI, it is easy to find a polygon enclosing it or its convex hull. To determine whether a sample point is close enough to the ROI to be useful, we determine whether it is inside the polygon using the winding number. If not, then we determine its distance to the boundary by projection to each line segment of the boundary. The winding number also provides a robust answer to determining if the point lies in the convex hull. (These same operations are used to determine if a coverage point is at least ε inside a

footprint.)

2.2.5 Optimization problem Q

The optimal set of footprints for Q is an epsilon approximation to P, in that any coverage of the same ROI with footprints that are smaller by epsilon must use at least the same number of footprints. The converted problem Q has a number of rows (constraints and objectives) linear in the number of coverage plus placement points. Each row has at most a linear number of columns (variables with non-zero coefficients), but usually much less depending on the choice of epsilon compared to the footprint size.

Despite this, solving the discrete optimization problem may have high complexity in theory and long runtime in practice. The number of rows and columns of Q are each $O(1/\varepsilon^2)$, leading to an $O(1/\varepsilon^4)$ dependence.

It is clear that in this conversion from P to Q that the combinatorial description of the boundary of the ROI and footprint is lost. The runtime of solving Q dominates, which is why we claimed that the method is insensitive to the ROI and footprint number of sides, holes, curvature, and convexity, etc.

We describe the optimization problem using Pyomo [19]. We have it call either CPLEX [6] (commercial license) or Gurobi [22] (free academic use license) as the underlying solver.

2.3 Describing and Solving the MIP Optimization Problem

Instances of the optimization problem Q were written in Python using libraries from the Pyomo open-source optimization modeling language. To describe the problem, we introduce several definitions. Let T denote the set of potential footprint placement points and C the set of required coverage points. For efficiency we usually choose sampling radii to keep $|T| \approx |C|$. We let $\delta_t \in \{0,1\}$ with $t \in T$ represent whether or not a footprint has been placed at placement point t. The function $g(c) : C \to J$, where $J \subset T$ and $|J| \geq 1$ is used to construct a constraint ensuring that all coverage points are covered by placed footprints:

$$\sum_{j}^{g(c)} \delta_j \geq 1 \, \forall c \in C. \tag{2.1}$$

The problem is further constrained in that at most one footprint may be placed at any given placement point:

$$\delta_t \leq 1 \, \forall t \in T. \tag{2.2}$$

The constraint in Equation 2.2 is redundant when defining the decision variables δ_t as binary in this version of Q. However, it is conceivable that future extensions of Q may require or benefit

from using a similar constraint. For example, one may extend Q to place multiple footprint shapes, or use additional constraints and relax the decision variables to take integer values. We include Equation 2.2 here for reference for understanding the corresponding Python script *geo_cover.py*. To ensure that all coverage points are covered by a minimal number of placed footprints, the objective function and corresponding mixed-integer linear program are:

$$\begin{aligned} \min \quad & \textstyle\sum_t^T \delta_t \\ \text{s.t.} \quad & \textstyle\sum_j^{g(c)} \delta_j \geq 1 \quad \forall c \in C, \\ & \delta_t \leq 1 \quad \forall t \in T, \\ & \delta_t \in \{0,1\} \quad \forall t \in T. \end{aligned} \tag{2.3}$$

2.3.1 Spreading Out Footprints

We explored a variation that discouraged placing footprints that strongly overlap, by adding constraints and objectives to the mixed-integer linear program of Equation 2.3. The idea is to count and minimize how many coverage points two footprints have in common. To make this precise, several additional definitions are necessary. The number of footprint placement pairs considered strongly affects Q's problem size, and therefore solve time. Hence we take care to only introduce variables for placement pairs that have non-zero overlap. We define a placement pair (i, j) where $i, j \in T$, over the set D of non-zero overlapping placement pairs, where $|D|$ depends on placement point locations and footprint shape. Let $f(i, j) : D \rightarrow [0, 1]$ be the mapping of placement pairs to their respective overlap. We then define $\phi_{i,j} \in \{0, 1\}$ with $(i, j) \in D$ to represent whether or not the placement pair (i, j) has footprints placed at both placement pair locations $i, j \in T$. To enforce that $\phi_{i,j} = 1$ if and only if $\delta_i = \delta_j = 1$, we construct the following three constraints to form a logical AND:

$$\phi_{i,j} \geq \delta_i + \delta_j - 1, \forall (i, j) \in D, \tag{2.4}$$

$$\phi_{i,j} \leq \delta_i, \forall (i, j) \in D, \tag{2.5}$$

$$\phi_{i,j} \leq \delta_j, \forall (i, j) \in D \tag{2.6}$$

The objective function of 2.3 is summed with an additional term penalizing overlap:

$$\begin{aligned} \min \quad & \textstyle\sum_t^T \delta_t + \sigma \sum_{(i,j)}^D f(i,j)\phi_{i,j} \\ \text{s.t.} \quad & \textstyle\sum_j^{g(c)} \delta_j \geq 1 && \forall c \in C, \\ & \delta_t \leq 1 && \forall t \in T, \\ & \delta_t \in \{0,1\} && \forall t \in T, \\ & \phi_{i,j} \geq \delta_i + \delta_j - 1, && \forall (i, j) \in D, \\ & \phi_{i,j} \leq \delta_i, && \forall (i, j) \in D, \\ & \phi_{i,j} \leq \delta_j, && \forall (i, j) \in D \end{aligned} \tag{2.7}$$

In the formulation of 2.7, σ is a constant used to scale the relative weighting of the term representing placed footprint overlap, $\sum_{(i,j)}^D f(i, j)\phi_{i,j}$. In practice, we chose σ to be quite small, $\sigma \approx 0.001$ to ensure that this term does not dominate the objective function relative to the integer valued term $\sum_t^T \delta_t$.

2.3.2 But Do Not Spread Footprints Far Beyond the Domain

The problem we observed when spreading out footprints in Section 2.3.1 was that we tended to place footprints that were mostly outside the ROI. Hence, we countered the spreading objective by a small term rewarding footprint placements that covered more coverage points. The objective function of 2.7 then became:

$$\sum_{t}^{T} \delta_t (1 - \sigma_1 h(t)) + \sigma_2 \sum_{(i,j)}^{D} f(i,j)\phi_{i,j} \tag{2.8}$$

where $h(t) : T \to \mathbb{N}$ is a mapping of the number of coverage points covered when a footprint is placed at a given placement point. This mapping is computed after the placement point sampling step and according to the footprint shapes used. As before, both σ_1 and σ_2 are chosen so as to allow the number of placed footprints to dominate the value of the objective function.

2.3.3 Other Footprint Objectives

It is worth mentioning that depending on the application at hand, it may instead be desirable to maximize overlap, for example, to increase redundant imaging in important regions or to assist in a stitching of separate camera images. The given formulations should be explored further to consider objective functions and constraints beneficial to alternative applications. In general, problem instantiations in which a minimal covering of less than ten footprints could be achieved were studied. Accordingly, solve times with either the CPLEX of Gurobi mixed-integer linear program branch-and-bound solvers found optimal solutions after running for between 30 seconds and three minutes on a machine with 32 cores and 1TB of RAM.

2.4 Multiple Footprint Shapes

In our formulation, it is straightforward to extend to multiple footprint shapes. These may come from different cameras, or from the same camera at different locations and view angles of the earth. We may have a budget of the number of times we can use each shape, or weight the use of each one in the objective. The differences are that we must calculate coverage for each footprint shape. Also, instead of one 0–1 variable for each placement point, we have a 0–1 variable for each placement point and footprint shape combination. We have not implemented this extension, but it does not appear difficult.

2.5 Faster Coverage Computation and Smarter Placement Points

In our implementation, for each placement point we compute which coverage points are covered independently by brute force. This is adequate for our setting because the problems are small, and

the overall time is dominated by the Q MIP solver time. However, it is possible to simultaneously compute a reasonable set of placement points, and which coverage points are covered for each of them, and this efficiency may be helpful for other settings.

The reference footprint is shrunk by ε and inverted through each of the coverage points: the inversion shape is the set of locations of the shrunk footprint's anchor point such that the shrunk footprint will cover that coverage point. The arrangement of intersections of all inversions are processed by a plane sweep, to find the cells that cover a maximal set of coverage points, the maximal canonical sets. For each such set, we may select a placement point.

Using inversion and planesweep to create the intersection cells and the maximal canonical sets is described in an arXiv paper, "Discretization of Planar Geometric Cover Problems" [23]. The problem they consider is how to identify the placements of a footprint that covers a maximal (i.e. a geometric local maximum) number of specified points. Inversion is a straightforward reflection of the footprint through the anchor point. We get to pick the anchor. For disk footprints, a natural choice is the disk center, so the inverted disk is just the disk itself. For polygons, a natural choice is the centroid. Note inversion preserves convexity. Plane sweeping the arrangement is standard. Sweep the arrangement and identify the cells. Placing the anchor point of the footprint anywhere inside a cell covers the same set of sample points. The maximal canonical sets are the cells who have no adjacent cells that cover more points. This can be discovered by a directed graph from low to high adjacent cells: maximal canonical cells have no outgoing edges.

2.6 Alternative Continuous Quadratic Optimization

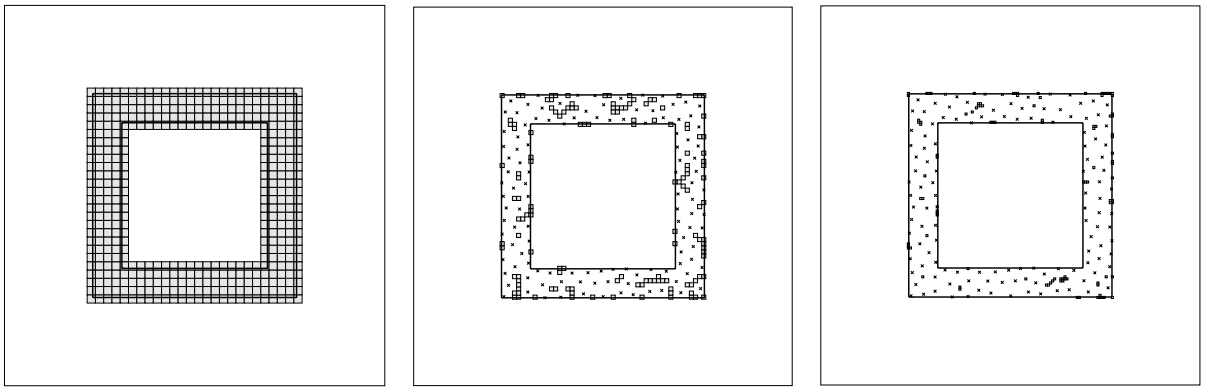
There is a continuous form of the problem for circular footprints. The following optimization problem tells you whether there is a placement of k footprints that covers all the sample points. So, the overall problem would be to find the smallest k for which the optimization problem is feasible. The decision variables are the (x, y) coordinates of each of the k footprints' reference points, the circle centers, denoted c_k. The optimization problem has no real objective, and just the constraint that for each sample point, the distance between it and the closest footprint is less than zero, i.e. it lies inside a footprint. The tricky part here is defining the distance. For circular footprints, the distance is just the distance to the center minus the radius, which is a simple quadratic if we use the square of the distance instead. However, for more complicated footprints, even something as simple as a convex square, the distance is not even a quadratic, and the distance to the footprint boundary can not be described by only the Euclidean distance to the footprint center.

2.7 Examples

Here we provide some example outputs.

2.7.1 Square Annulus ROI, Circular Footprint

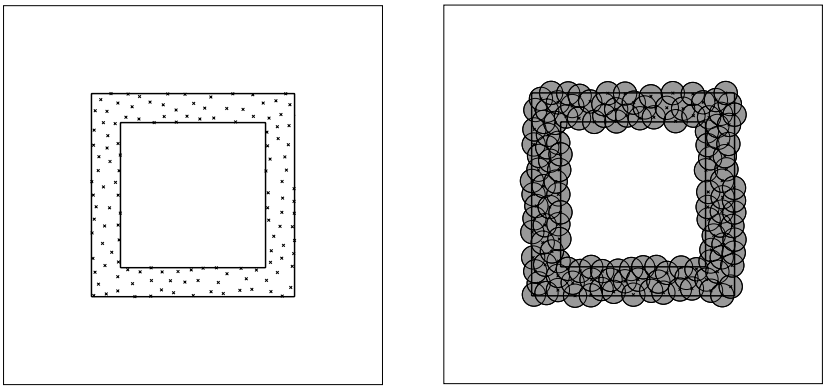
See Figures 2.1–2.4. Placement samples are ε apart, and for coverage samples, we used a larger distance, typically $\sqrt{2}\varepsilon$. Using this distance reduced the problem size considerably, with only a slight loss of solution quality. Since the ROI is non-convex, this problem illustrates the value of allowing footprints to be placed outside the ROI.



(a) Background grid of squares containing the ROI.

(b) End of iteration 1

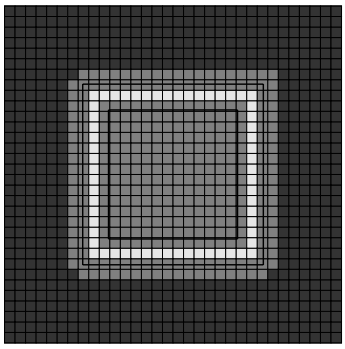
(c) End of iteration 2



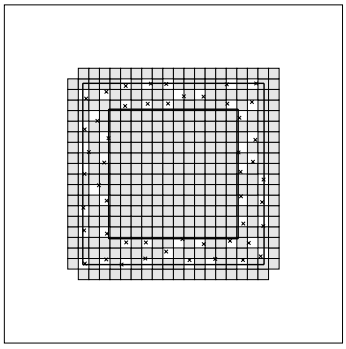
(d) Final Coverage Points. Any uncovered square is smaller than machine precision.

(e) The union of ε-radius circles at coverage points covers the entire ROI.

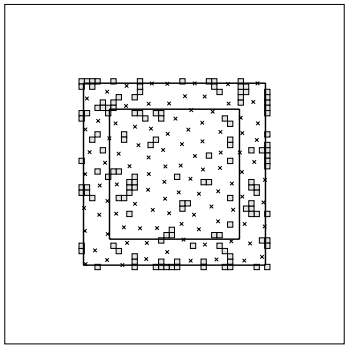
Figure 2.1. Creating the *coverage* samples for a square torus ROI. Placement points are "x" and the small squares are the refined background grid cells that are not yet covered by an ε-radius circle centered at a placement point. Here we have 134 placement points.



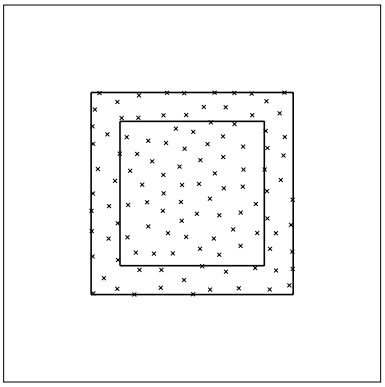
(a) Background grid of squares containing the ROI, expanded by the footprint radius and restricted by the convex hull. White squares are strictly interior, and the darkest squares are invalid.



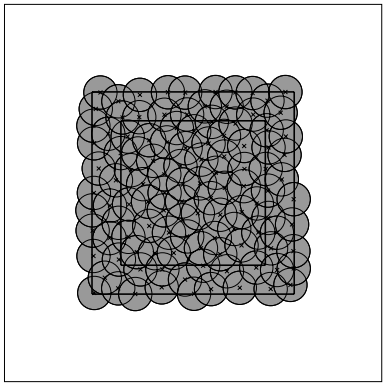
(b) The placement samples are initialized by a maximal subset of the coverage points, limited by the larger intersample placement distance.



(c) End of iteration 1



(d) Final Placement Points. Any uncovered square is smaller than machine precision.



(e) The union of circles at placement points covers the expanded ROI.

Figure 2.2. Creating the *placement* samples for a square torus ROI using $\sqrt{2}\varepsilon$ spacing. The placement points can be outside the ROI, up to the footprint radius distance, but are limited to the convex hull to keep the problem size small. Here we have 113 placement points.

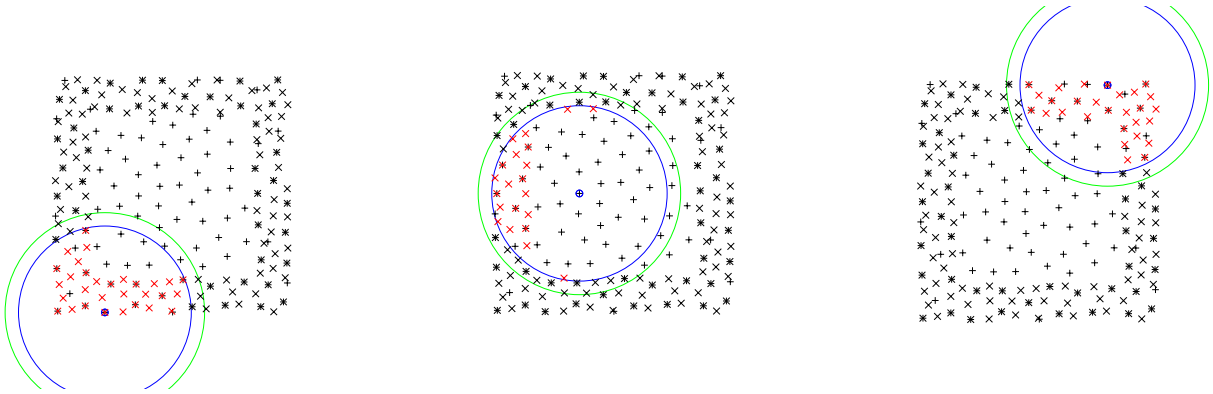


Figure 2.3. Calculating the coverage points that are covered by centering a footprint at a placement point. Placement points are "+" and coverage points are "x". The footprint is the green circle. The footprint radius is reduced by ε to the blue circle, in order to ensure that any gaps between the coverage points in the ROI are covered by the green circle. Coverage points inside the blue circle are drawn in red and included in the MIP Q to model "if a footprint is placed here, these coverage points are covered."

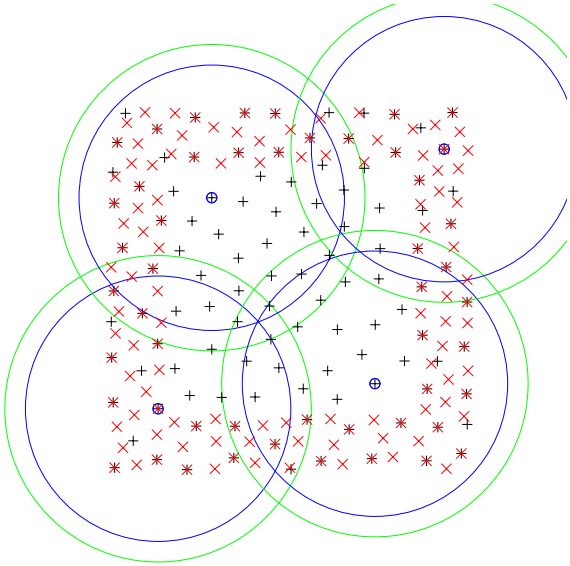
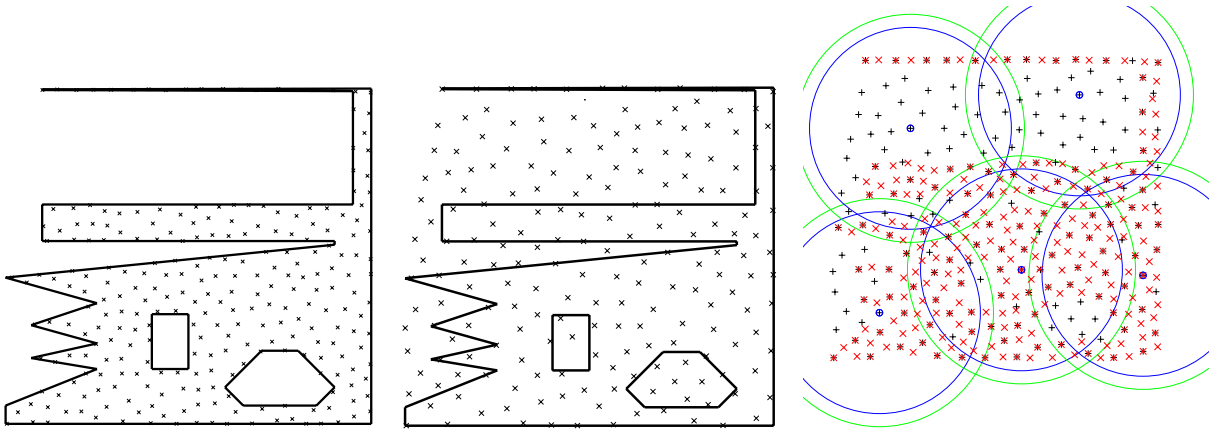


Figure 2.4. The optimal solution to this MIP Q uses four footprints, including two that are outside the ROI. All coverage points are red, reflecting the constraint that every coverage point must be covered by at least one placed footprint.

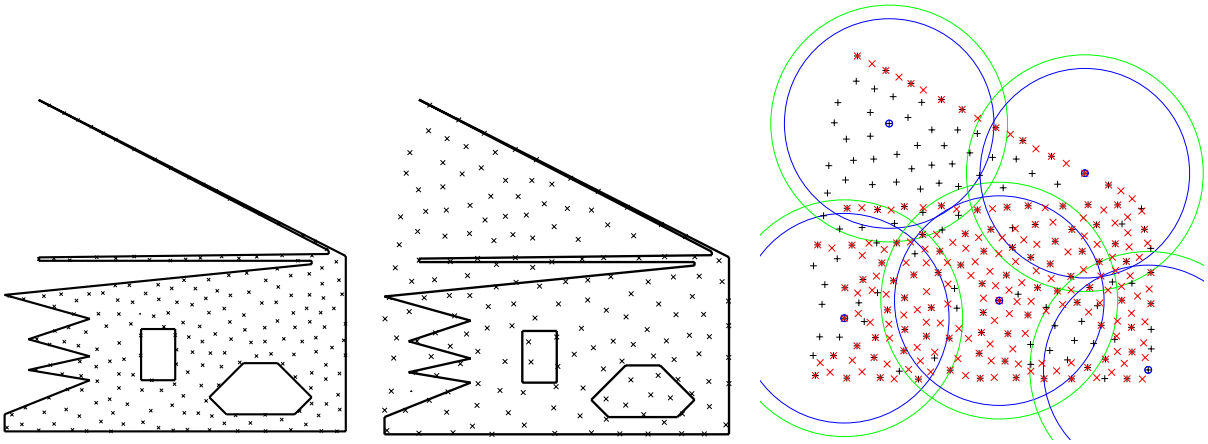
2.7.2 Additional Circular Footprint Examples

Figures 2.5–2.7 illustrate a circular footprint placed on some additional ROI shapes. The large "Mopad" example in Figure 2.8 has about 800 coverage points, 600 placement points, and an optimal solutions with 23 footprints that takes about 10 minutes for CPLEX to solve. The solver performance does not directly scale with the problem size, and largely depends on things like the number of local minima and other abstract optimization problem structure, and is hard to predict for new problems.



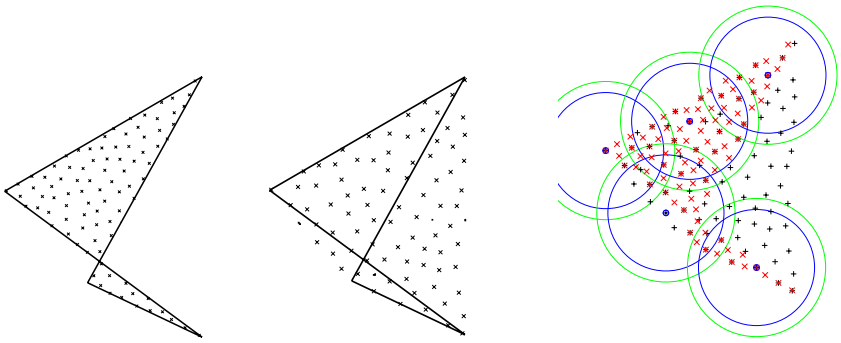
(a) Coverage **(b)** Placement **(c)** Solution

Figure 2.5. "Sharp" ROI example.



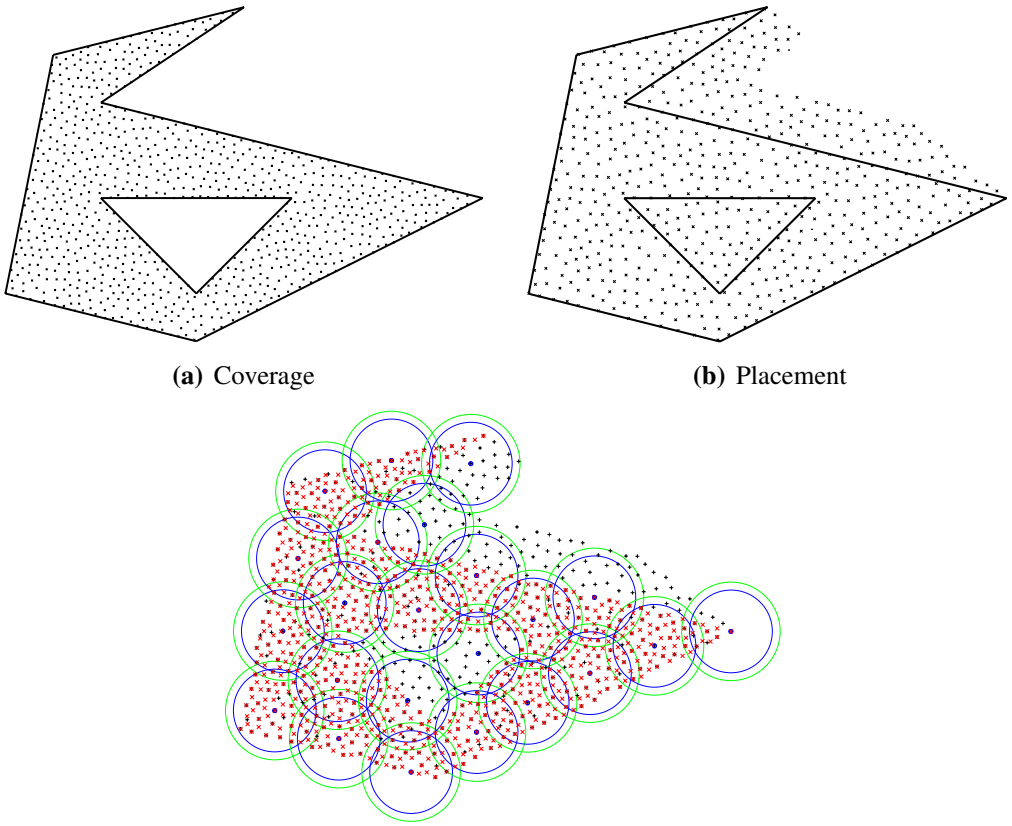
(a) Coverage **(b)** Placement **(c)** Solution

Figure 2.6. "Sharp-diagonal" ROI example.



(a) Coverage **(b)** Placement **(c)** Solution

Figure 2.7. "Arrow" ROI example.



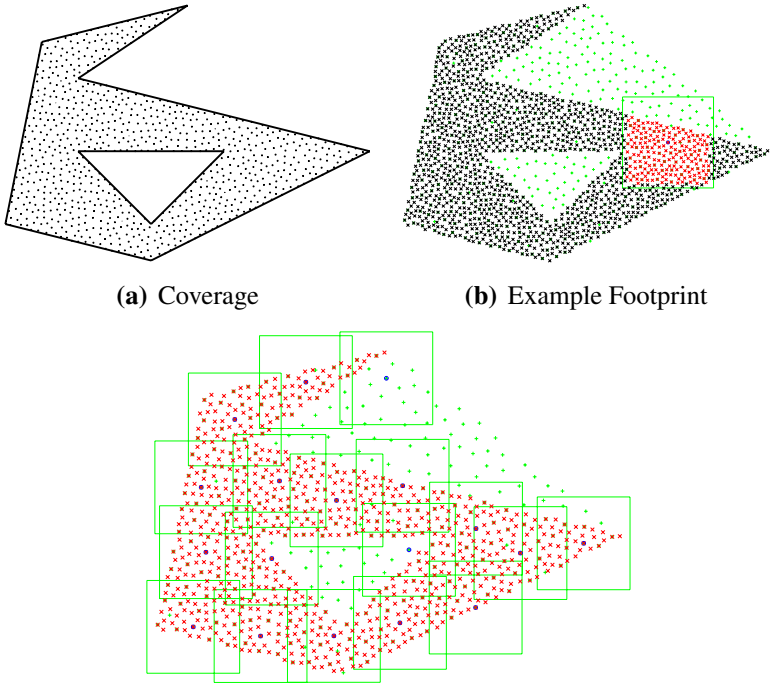
(a) Coverage **(b)** Placement

(c) Solution

Figure 2.8. "Mopad" ROI example.

2.7.3 Non-circular Footprint Examples

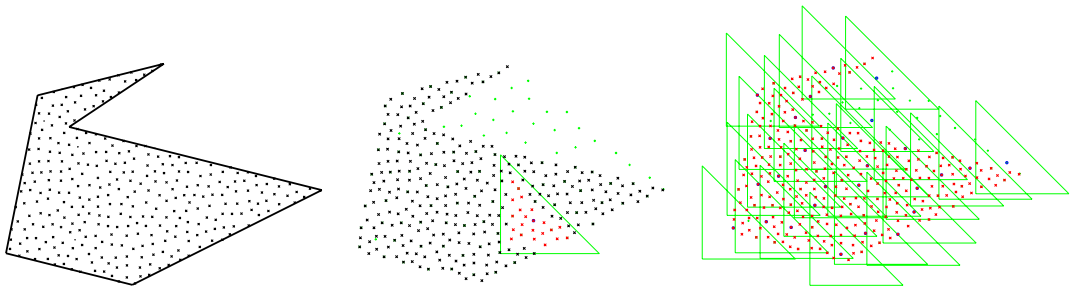
Figures 2.5–2.7 illustrate placing non-circular, polygonal footprints. The "Right Triangle" example in Figure 2.10 illustrates the well-known observation that a right-triangle, without rotations, is an inefficient covering shape. The "Projected Square" footprint in Figure 2.12 is an example of the shape of a part of the globe seen by a square camera lens aimed at a glancing angle. The ROI and footprint are projected to the plane for the purposes of our algorithm.



(a) Coverage **(b)** Example Footprint

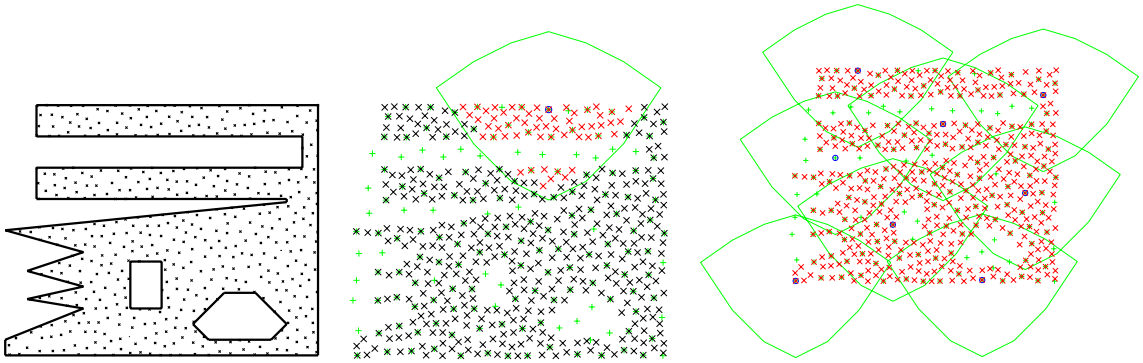
(c) Solution

Figure 2.9. "Square" polygon footprint example.



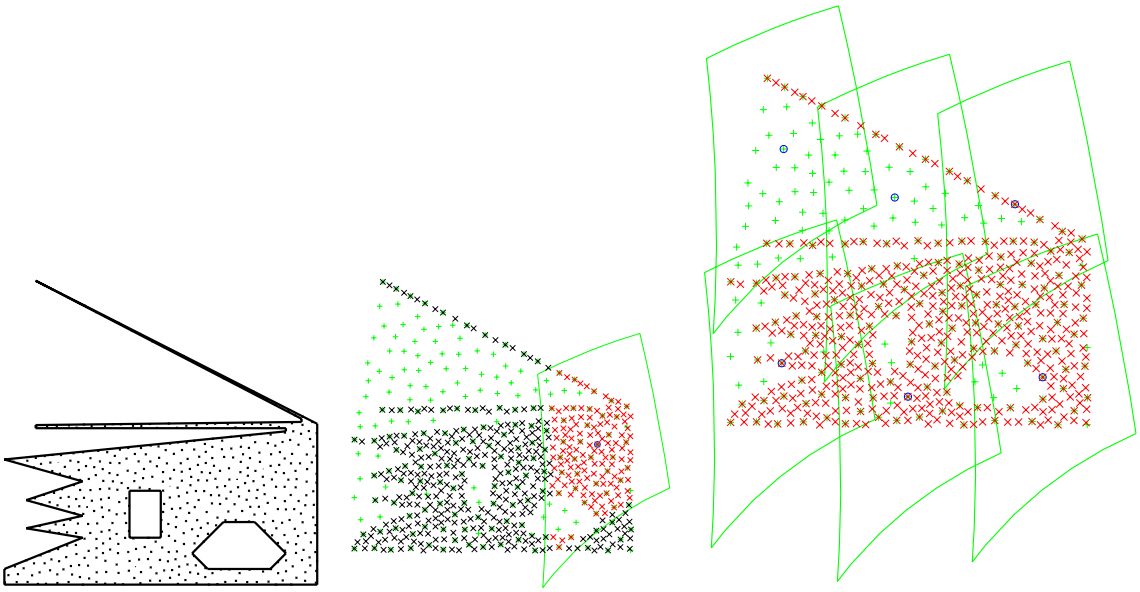
(a) Coverage **(b)** Example Footprint **(c)** Solution

Figure 2.10. "Right Triangle" polygon footprint example.



(a) Coverage **(b)** Example Footprint **(c)** Solution

Figure 2.11. "Egg" polygon footprint example.



(a) Coverage **(b)** Example Footprint **(c)** Solution

Figure 2.12. "Projected Square" polygon footprint example.

Chapter 3

Sub-footprint Placement Optimization

We study a bandwidth constrained remote sensing problem. In our problem of interest, we consider a system in which full frame images are too bandwidth expensive to produce at a high frame rate. To ameliorate the bandwith constraint, the image is subidivided up into rectangular regions, *sub-footprints*, that can be more efficiently sent downstream for processing. Our objective is to optimally choose rectangular sub-footprints that both cover regions of particular interest while not exceeding the bandwidth constraint. Our approach is to characterize the problem as a mixed integer linear constrained optimization problem which can be efficiently solved using a state of the art solver.

We consider an image of $M \times N$ pixels that is further subdivided up into fixed size subsets called *chips*. A chip is a $m \times n$ array of pixels such that there are $M_C \times N_C$ total chips in the image (where $m \cdot M_C = M$ and $n \cdot N_C = N$). In the model we are studying, the sub-footprint is comprised of a rectangular array of chips. We assume the bandwidth allocated to each sub-footprint is equal, regardless of the overall size of the sub-footprint. Consequently, if there were two sub-footprints, one with 4 chips and one with 8 chips, each sub-footprint would be allowed $\frac{B}{2}$ bandwidth, so the chips in the first sub-footprint would get $\frac{1}{4} \cdot \frac{B}{2}$ bandwidth and in the second sub-footprint, each chip would receive $\frac{1}{8} \cdot \frac{B}{2}$ bandwidth per chip. The total number of sub-footprints is constrained to be a fixed, maximum number.

Each pixel has a corresponding *priority* value. We define the priority of a chip to be the sum of the priorities of its corresponding pixels. Similarly, the priority value of a sub-footprint containing several chips is the sum of the priorities of its corresponding chips. Our goal is to optimally choose sub-footprints that capture the highest priority chips while constrained by the bandwidth constraint. A sub-footprint could be chosen to cover many chips, at the disadvantage that the bandwidth per chip is very low. On the other hand, one could choose small sub-footprints to cover chips of interest, but since their are a fixed number of sub-footprints that can be sent downstream, this reduces the total priority of the sub-footprints sent.

3.1 Problem Formulation

The image is defined as a Cartesian grid of $M \times N$ pixels, which is sub-divided up into a coarser grid of $M_C \times N_C$ chips (each comprised of $m \times n$ pixels). The chips are a fixed partition on the

image plane, and cannot move. We let S_k denote the k-th sub-footprint, which is a connected, rectangular set of chips, and we define $C_{i,j}$ to refer to the chip in the ith row and jth column of the image. We assume there is a fixed, maximum number of possible sub-footprints that can be chosen. We denote the priority of chip $C_{i,j}$ as $p_{i,j}$. We introduce an indicator function $\chi_{i,j}^k$ where

$$\begin{cases} \chi_{i,j}^k = & 1 \text{ if } C_{i,j} \in S_k, \\ & 0 \text{ otherwise.} \end{cases} \tag{3.1}$$

The area of a sub-footprint corresponds directly to the number of chips contained within it. Consequently, we may define the number of chips in a sub-footprint (or the *area* of a sub-footprint) $A_k = \sum_{i,j} \chi_{i,j}^k$.

Given a bandwidth B, and k sub-footprints, each sub-footprint gets $\frac{B}{k}$ total bandwidth. We prefer to give higher priority chips more bandwidth; consequently, if two solutions both include a chip C with a high priority, we prefer the solution where C is in the smaller of the two sub-footprints. This implies the larger sub-footprints tend to have the lower priority chips. We can enforce in our objective function a penalty term proportional to the area of the sub-footprint. Our goal is to devise an optimization model that tries to simultaneously cover high priority chips, give high priority chips the most bandwidth possible, and guaranteed that the bandwidth constraint is enforced.

3.2 Chip Position Constraints

We need to enforce that sub-footprints do not intersect each other. That is, if $C_{i,j} \in S_k$, then $C_{i,j} \notin S_l$ for $l \neq k$. For each sub-footprint, we can define variables to keep track of the upper left and lower right coordinates of the sub-footprint in chip space. These two locations are sufficient to uniquely characterize the sub-footprint since we define sub-footprints to be rectangular arrays of chips. We assume that all chips are rectangles of the same size. We let Δx denote the length of a chip in the x direction in units of pixels and Δy denote the length of the chip in the y direction in units of pixels.

We begin by enforcing a basic constraint that the sub-footprints lower left corner is always to the left of its upper right corner. Let S denote a sub-footprint and let S_0 and S_1 denote the lower left and upper right coordinates of the sub-footprint in chip space. Let $S_0(x)$ denote the x coordinate of S_0 and $S_0(y)$ denote the y coordinate of S_0. We enforce

$$S_0(x) \leq S_1(x) \tag{3.2}$$

to guarantee that S_0 is to the left of S_1. Similarly, we enforce

$$S_0(y) \leq S_1(y) \tag{3.3}$$

to guarantee that the upper right coordinate of the sub-footprint is to the top right.

We need to keep track of which chips are covered by a sub-footprint. For a fixed footprint S and chip C, we define four variables V_1, V_2, V_3, and V_4 to track where chip C is with respect to the

corners of S. We let V_1 be an indicator variable which is 1 if C is to the right of the lower left corner of S, we let V_2 indicate if C is above the lower left corner, V3 indicates if C is to the left of the upper right corner of S, and $V4$ indicates if the chip is below the upper right corner. A chip C is in sub-footprint S if $V_1+V_2+V_3+V_4=4$. For chip $C_{i,j}$ and sub-footprint S_k, the indicator variable $\chi^k_{i,j}$ should be true only if $V_1+V_2+V_3+V_4=4$, and therefore, we add the constraint

$$\chi^k_{i,j} \geq V_1+V_2+V_3+V_4-3. \tag{3.4}$$

The values of V_i are determined by the position of the chip C with respect to the sub-footprint S. $V_1=1$ only if the chip is to the right of the lower left corner, so we define the constraint

$$S_0(x)-(1-V_1)\cdot\Delta x \leq C_0(x) \tag{3.5}$$

where Δx is the x length of the chip C and $C_0(x)$ is the lower left coordinate of the chip. From (3.5), the only way for $V_1=1$ is if $S_0(x)\leq C_0(x)$, as desired. Similarly, $V2$ is true if

$$S_0(y)-(1-V_2)\cdot\Delta y \leq C_0(y). \tag{3.6}$$

The upper right corner conditions are constrained in a similar way. We enforce V_3 by constraining

$$S_1(x)-(\Delta x+1)\cdot V_3 \leq C_0(x)-1. \tag{3.7}$$

and V_4 is constrained by

$$S_1(y)-(\Delta y+1)\cdot V_4 \leq C_0(y)-1. \tag{3.8}$$

Equation (3.4) guarantees that $\chi^k_{i,j}=1$ provided the $C_{i,j}\in S_k$. However, the aforementioned constraints do not prevent $\chi^k_{i,j}\geq 1$ for a chip outside of S_k. To make $\chi^k_{i,j}$ an absolute indicator of whether $C_{i,j}$ is or is not in S_k, we further constrain $\chi^k_{i,j}$ with respect to the vertex V rules. We can achieve this by enforcing that $\chi^k_{i,j}$ is bounded above by each of the vertex rules. We enforce:

$$\chi^k_{i,j} \leq V_\ell \text{ for } \ell=1,2,3,4. \tag{3.9}$$

With the vertex rules along with (3.9), it follows that $\chi^k_{i,j}=1$ if and only if $C_{i,j}\in S_k$.

Enforcing a logical ordering on the sub-footprints provided significant speed-up to the time to solution. We lexicographically order the sub-footprints with the first sub-footprint in the lowest left position possible. The lexicographic ordering of the sub-footprints would not change the actual overall solution, but guarantees a specific ordering only. Let S_k and S_{k+1} denote two sub-footprints. Recall that a lexicographic ordering of ordered pairs implies that for $(x_0,y_0)\leq(x_1,y_1)$ if and only if $x_0<x_1$ or $x_0=x_1$ and $y_0\leq y_1$. To ensure that S_k is in a position at least lower and at least not to the right of S_{k+1}, we can constrain

$$S^k_0(x)+\Delta x\cdot S^k_0(y) \leq S^{k+1}_0(x)+\Delta x\cdot S^{k+1}_0(x). \tag{3.10}$$

where we recall that x specifies the column position and y specifies the row position.

3.3 Area and Bandwidth Constraints

The area of the sub-footprint is computed as the number of chips within the sub-footprint. The area of sub-footprint S_k, denoted A_k, is important for bandwidth considerations. Given a per sub-footprint bandwidth B_k, the bandwidth per chip is $\frac{B_k}{A_k}$. Consequently, larger area sub-footprints get smaller bandwidths per chip, which may be disadvantageous if there exists high priority chips contained within the sub-footprint.

We track the area by introducing an area variable A_k per sub-footprint S_k. Recall that a chip $C_{i,j} \in S^k$ if and only if $\chi^k_{i,j} = 1$. Consequently, the area of the chip can be computed by summing all indicator variables χ^k over each chip. We therefore introduce the variable in the optimization problem formally by

$$A_k = \sum_{i,j} \chi^k_{i,j}. \tag{3.11}$$

The bandwidth requirements constrain that each sub-footprint receives equal bandwidth. We additionally have that the bandwidth constrains the total number of chips that can be downlinked. Since the number of chips per sub-footprint is controlled by the area A_k, we have that $\sum_{k \in K} A_k \leq B$ which controls the total number of chips that can be sent. We may further enforce a maximum area of a single sub-footprint. Let $\|A\| = \max_k A_k$ denote the maximum of the areas of the sub-footprints. We enforce a maximum area by the constraint

$$K \cdot \|A\| \leq B \tag{3.12}$$

which prevents sub-footprints from exceeding the bandwidth constraint.

3.4 Objective Function

We now introduce the objective function for the optimization problem which we seek to maximize. We discuss different objective functions for maximization purposes and the reasoning behind our consideration of them.

At the most basic level, the sub-footprint problem should seek to maximize the number of high priority chips covered by sub-footprints. A *reward term* captures the benefit of covering a chip with a sub-footprint can be computed by

$$R = \sum_{k \in K} \sum_{i,j} \chi^k_{i,j} \cdot p_{i,j}. \tag{3.13}$$

The reward term is computed by summing, for each sub-footprint, the binary indicator variables for whether a chip is covered by the sub-footprint or not times the corresponding priority of the covered chip. Consequently, each chip that is covered increases the value of the reward term.

Due to the fixed bandwidth per sub-footprint, it would be desirable if higher priority chips were captured in smaller sub-footprints. Since the bandwidth per chip is smaller for large sub-footprints, it would be advantageous to penalize the objective function value in the event that high priority chips are grouped inside large sub-footprints. For each instance that a chip is covered by a sub-footprint, one approach is to subtract an *penalty term*. One option for a penalty term is

$$P_1 = \varepsilon \sum_k \sum_{i,j} A_k \cdot p_{i,j} \tag{3.14}$$

where ε is a user chosen parameter. For each chip that is covered, the penalty term increases if the area of the sub-footprint covering the chip is large. Consequently, for large $p_{i,j}$, we should prefer it is placed a sub-footprint with a smaller A_k. For equal total sub-footprint priority, smaller sub-footprints consequently provide a smaller penalty.

We furthermore need to prevent zero priority chips from being covered by a sub-footprint if possible. To ensure this, we may subtract another penalty term for zero priority chips simply defined as

$$P_2 = \gamma \sum_{k,i,j} \chi_{i,j}^k \tag{3.15}$$

where γ is a user specified penalty parameter.

With the reward and penalty terms, we introduce a first idea for an objective function by taking the difference between the reward and penalty terms. We define the objective function F as

$$F = \sum_{k \in K} \sum_{i,j} \left(\chi_{i,j}^k \cdot p_{i,j} - \varepsilon A_k \cdot p_{i,j} \right) - \gamma \sum_{k,i,j} \chi_{i,j}^k \tag{3.16}$$

where we recall that $p_{i,j}$ denotes the priority of chip $C_{i,j}$.

However, this suffers from the error that the penalty term (3.14) is subtracted for each $\chi_{i,j}^k$ regardless of whether or not $\chi_{i,j}^k$ is one or zero. However, this intuitively expresses the objective function we aim for. We instead introduce real valued variables to allow us to turn off the penalty in the case that $\chi_{i,j}^k = 0$.

We introduce a variable $Z_{i,j}^k$ that is nearly equal to $\chi_{i,j}^k$ except slightly penalized by the size of the respective footprint A^k, but zero in the case that $\chi_{i,j}^k = 0$. Consequently, the new reward term would be

$$\tilde{R} = \sum_k \sum_{i,j} Z_{i,j}^k \cdot p_{i,j}. \tag{3.17}$$

In order for Z to be nearly χ, but slightly penalized, we allow Z to be a real variable rather than a binary indicator variable. Furthermore, we provide real-valued bounds between 0 and 1 for Z. We enforce a constraint that Z is bounded above by χ via

$$Z_{i,j}^k \leq \chi_{i,j}^k. \tag{3.18}$$

Note that this constraint enforces that if a chip is not in sub-footprint χ^k, $Z^k_{i,j}$ is forced to be zero.

We introduce a slack variable s that is guaranteed to be zero when χ is one. The slack variable is similarly bounded as a real variable between 0 and 1. The constraint relating χ and s is enforced by the rule

$$s^k_{i,j} \leq 1 - \chi^k_{i,j}. \tag{3.19}$$

Note that if $\chi^k_{i,j} = 1$, then $s^k_{i,j} = 0$. This implies that s is zero for chips inside of sub-footprint S_k. We penalty term is built into Z via the constraint that

$$Z^k_{i,j} - s^k_{i,j} \leq \chi^k_{i,j} - \varepsilon A_k. \tag{3.20}$$

The slack variable s enables us to turn off the penalty in the case that the chip is not covered in the footprint and it allows $Z^k_{i,j}$ to equal $\chi^k_{i,j}$ in the case that $\chi^k_{i,j} = 0$.

With this new model for representing covered chips with penalty terms, we may introduce our objective function. The Z terms times the respective chip priorities acts as a reward term, while we penalize sub-footprints for covering zero priority chips via the same penalty term defined in (3.15). We define

$$\tilde{F} = \sum_{k,i,j} Z^k_{i,j} \cdot p_{i,j} - \gamma \sum_{k,i,j} \chi^k_{i,j}. \tag{3.21}$$

For an example experimental result, please see Figure 3.1(b).

3.5 Heuristic Methods

We discuss a greedy heuristic method for generating sub-footprint placements. A heuristic method could provide significantly faster sub-footprint generation due to the low computational cost relative to an optimization scheme.

We propose a greedy heuristic method for choosing sub-footprints based on the idea of iteratively choosing sub-footprints that choose the largest priority. We begin by generating a collection of possible sub-footprint shapes by determining all possible rectangular arrays of chips that we do not exceed the maximal area of a sub-footprint per the bandwidth constraint.

Each sub-footprint is constructed by considering the different allowable sub-footprint shapes and placing them along the image by choosing the potential sub-footprint placement that has the maximal priority while not intersecting another existing sub-footprint. Consequently, the largest area sub-footprint is not always the shape chosen. Once a sub-footprint is placed, those chips are marked as covered and a subsequent sub-footprint cannot be placed that intersects covered chips. The process repeats to cover the image array with sub-footprints that maximizes the priority of the individually placed sub-footprints at each step.

The greedy solution is guaranteed to have a lower priority than the solution generated by the optimization scheme. However, the time to generate the greedy solution is nearly instantaneous in

contrast to the MIP runtime. Figure 3.1 displays the results of a greedy solution in Figure 3.1(a) and an optimization solution Figure 3.1(b) for the same chip layout problem. The optimization solution produces a significantly higher objective value as well as a distinctly different solution behavior.

3.6 Shifting Pixels to Find Optimal Sub-footprints

In Section 3.4, we presented a method for generating optimal sub-footprint solutions for a list of prioritized chips to cover. In this section, we explore an alternative idea of shifting the image plane to align the corresponding pixels more optimally into different chips, which consequently may yield more optimal sub-footprint solutions.

Given an image (e.g., a bitmap image of pixel values), we divide the image up into chips based on fixed boundary lines on the image plane. One way to determine the priority or importance of a chip is to have a set of desired pixels to be covered (e.g., a feature of interest in an image such as a river). Given a list of $M \times N$ pixels $\{\phi_{k,l}\}$ and a list of their corresponding priorities $\{\rho_{k,l}\}$, the chips partition the list of pixels disjointly. We can then define the priority of a chip, $p_{i,j}$ by summing over the corresponding pixel priorities for each pixel within chip $C_{i,j}$. That is, we define

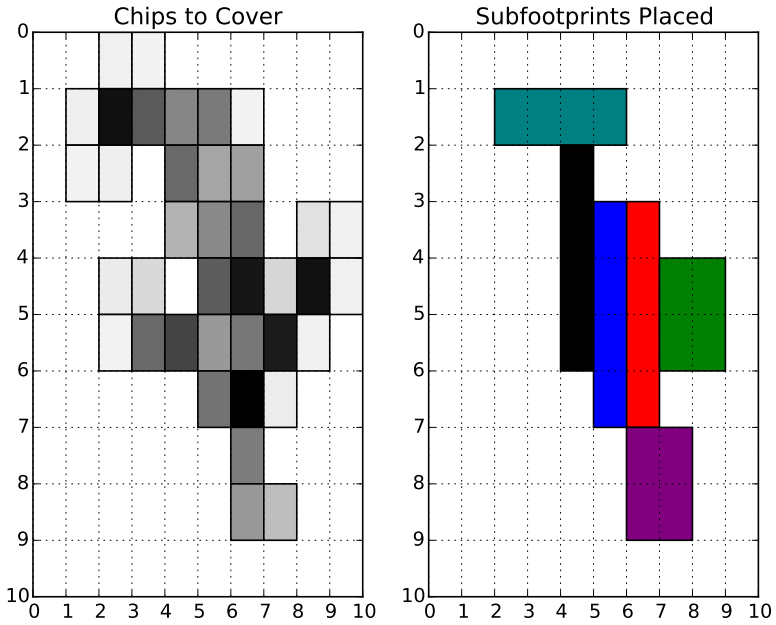
$$p_{i,j} = \sum_{\phi_{k,l} \in C_{i,j}} \rho_{k,l}.$$

This enables a user to define features of interest, and a method automatically converts an image scene into a list of chips to cover with corresponding priority values, which can then be optimized via the algorithm described in Section 3.4.

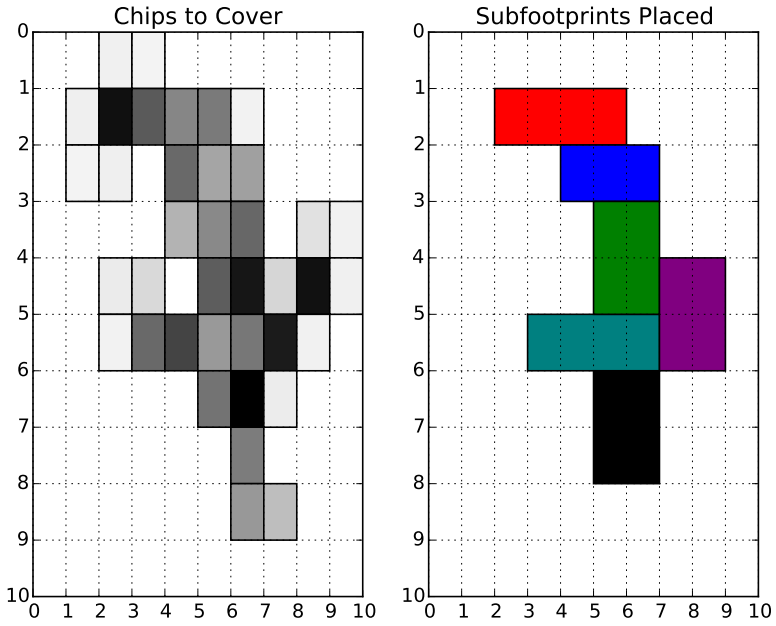
We consider the following question: if the image scene shifts by n pixels in the horizontal direction and m pixels in the vertical direction, the corresponding priority values of $p_{i,j}$ will shift (as the prioritized pixels may now lie within a different chip). Shifting the image plane may result in more concentrated, higher priority chips, which can lead to smaller sub-footprint placements.

Figure 3.2 displays an example of an input image with desired pixels corresponding to the Mississippi River. The objective is to optimally choose a sub-footprint covering given bandwidth limitations and a fixed number of possible sub-footprints. By shifting the image plane (i.e., reorienting the imaging device), we may be able to get a more optimal footprint covering. We consider an experiment with a maximum of 6 possible sub-footprints and a total bandwidth of value of 24. We assume the image is subdivided into an array of 10 chips by 10 chips with an input image of 160 pixels by 120 pixels. The darker pixels have higher pixel priorities, which leads to correspondingly higher chip priorities.

The chip priorities are determined by summing over the pixel priorities, and then the optimization algorithm is applied to the corresponding collection of prioritized chips to cover. To study the possible benefit of reorienting the image to generate a more optimal sub-footprint placement (by changing which prioritized pixels lie in which chips), we consider shifting the image vertically



(a) Greedy heuristic solution.



(b) Optimal MIP solution.

Figure 3.1. A greedy solution vs. an optimized solution for the same chip layout. The objective value of the 3.1(a) is 97.8% of the objective value of 3.1(b).

and horizontally to generate new prioritized chip layouts and solve for the resulting sub-footprint layout.

We experiment with the concept of pixel shifting and image plane re-orienting by starting with Fig 3.2 and shifting the image in the vertical and horizontal directions to generate many different possible chip layouts. We then apply the optimization algorithm for sub-footprints to each possible generated chip layout to empirically study whether or not there is a benefit to shifting the imaging plane to align chip boundaries in a desirable format. Figure 3.3 displays two of the possible outputs from the shifted image. The corresponding objective function values and generated sub-footprints are significantly different. Figure 3.3 demonstrates that shifting the imaging plane by pointing the imaging device appropriately can lead to improved sub-footprint generation with a significantly higher number of chips produced. Furthermore, some optimally generated sub-footprints possess qualitatively undesirable features such as gaps between sub-footprints covering a single region (as can be seen in Figure 3.3(a)). The results suggest that re-orienting the imaging plane can provide substantially improved, both quantitatively via the objective function value and qualitatively by sub-footprint placements, sub-footprint generation.

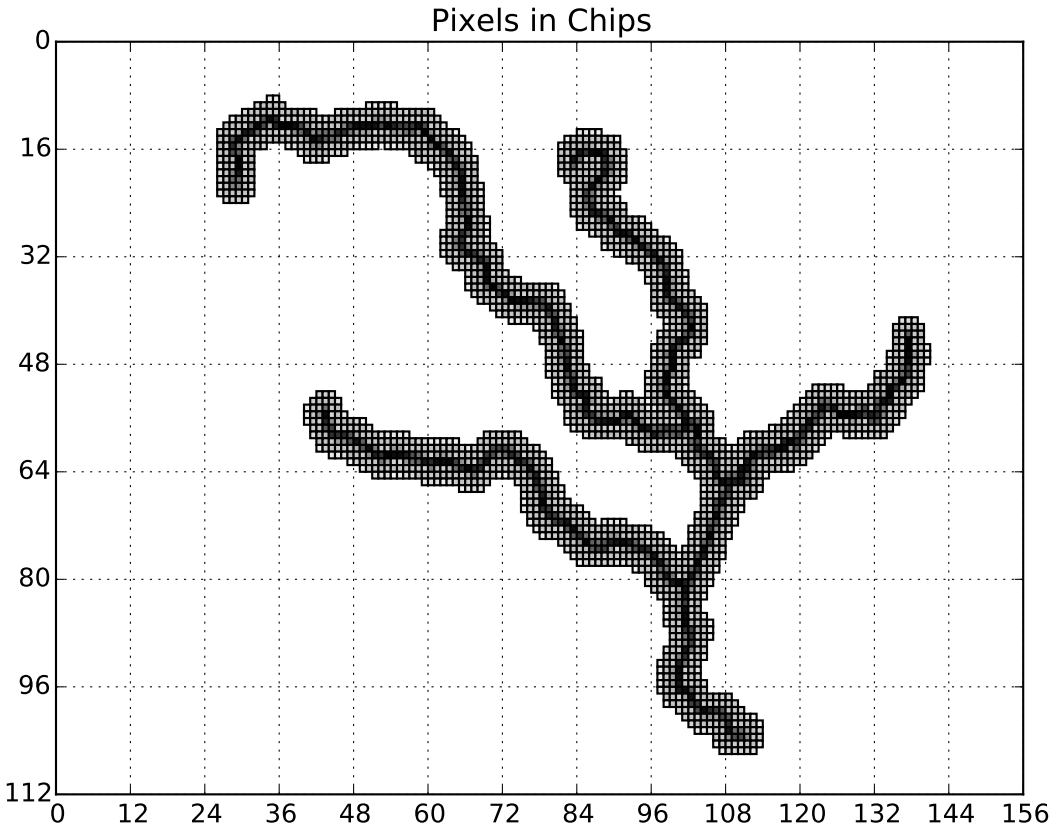
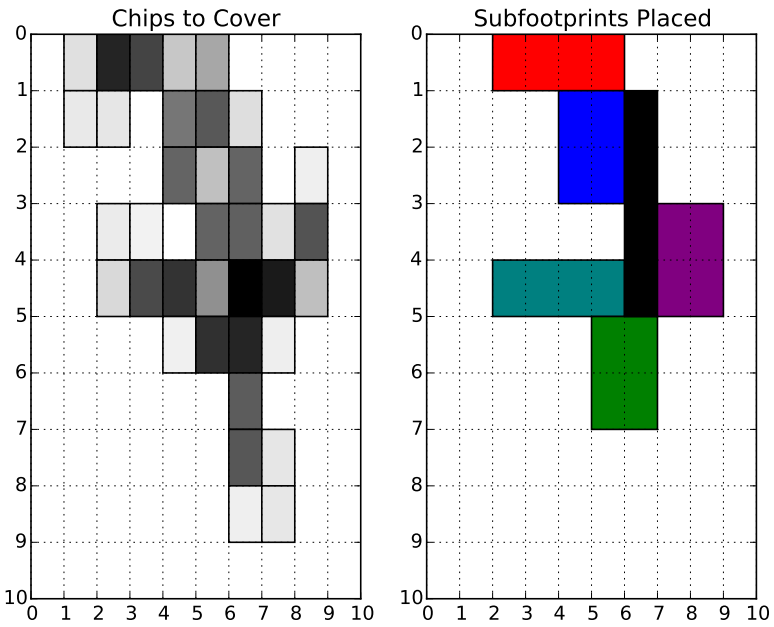
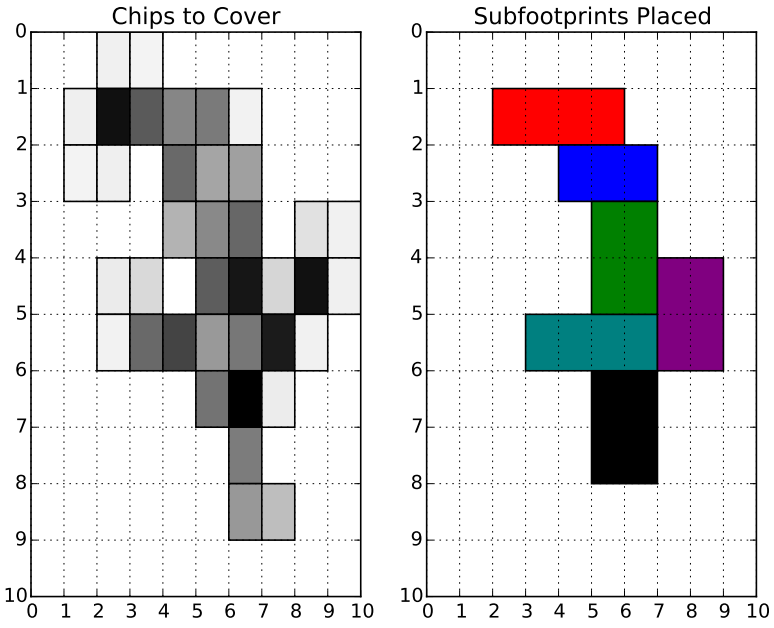


Figure 3.2. A representation of the Mississippi River. The grid lines indicate chip boundaries. Courtesy Wikipedia.



(a) A chip layout.



(b) The same layout shifted by a few pixels.

Figure 3.3. Two different MIP solutions for a shifted pixel image. The objective value of the 3.3(a) is 97.8% of the objective value of 3.3(b). Also note the qualitatively different chip layouts.

Chapter 4

Developed and Incorporated Software

Over the course of the project, several pieces of software were used and several new tools were developed. In the following section, we will outline where copies of the software can be obtained, what dependencies are required, and provide a brief overview of their capabilities. In general, scheduling and placement problems were instantiated and solved using a Sandia 64-core server with 2.4GHz AMD processors and 512GB (subsequently upgraded to 1TB) of RAM. The development environment was Linux and source code was primarily developed using Python 2.7 with additional code written in C++ and much of the web-based scheduler written using JavaScript.

4.1 Pyomo

All models developed for the collection scheduling and footprint/sub-footprint placement problems make use of Sandia's previously developed Pyomo [19] modeling language, version 4.0 or newer. See http://www.pyomo.org for more details. A wide variety of model and problem types are supported by Pyomo. The list includes nonlinear and quadratic programming as well as variations of mixed-integer programs, bilevel programming and more. Pyomo allows model and solution algorithm developers to develop scripts that can harness the many open-source packages the Python programming language provides. For example, we use `matplotlib` version 1.4.3 or newer to visualize resultant schedules. Pyomo may be installed in a few different ways, those familiar with the Python `pip` package can simply invoke `pip install Pyomo` if developing in a Linux environment with `pip` installed. Information regarding the size and runtimes associated with multiple collection scheduling models are given in Sections 1.2.4 and 1.2.10.

4.2 Solvers

4.2.1 Third-Party

Over the course of the project, the primary solvers used for all collection scheduling and placement algorithms were the CPLEX [1] and Gurobi [2] commercial solvers. Sandia has purchased multiple licenses of varying type for both solvers. For those interested, Gurobi also provides an Academic evaluation license for those conducting basic research in optimization. Additional third-party open-source solvers exist mixed-integer programs and other model types. A good resource for learning about other open-source tools for operations research is the Computational Infrastructure for Operations Research (COIN-OR) website http://www.coin-or.org/resources.html.

4.2.2 Collection Scheduling Heuristics

Texas A&M University collaborators developed heuristics for the deterministic model in Equation (1.6) and reported schedule quality and run-time statistics in [41]. These heuristics were written to solve models defined using the Julia programming language. This source code is available at https://gitlab.sandia.gov/82634/ops.

4.2.3 Sub-Footprint Heuristics

Heuristics for sub-footprint placement are included in the GeoPlace source code. They are available for download under an open-source license at https://github.com/cgvalic/GeoPlace. Please see Section 3.5 for more details about these heuristics.

4.3 Models, Scripts, and Applications

4.3.1 GeoPlace

The GeoPlace software libraries include several software tools for the footprint and sub-footprint placement problems. Specifically included in the GitHub repository are multiple Pyomo models for each of the two placement problems. Footprint placement model variations include an objective to minimize placed footprints and to additionally minimize overlap. Sub-footprint variations include constrained bandwidth placement and an objective to maximize average bandwidth for placed sub-footprints. For the footprint placement model, C++ libraries for domain sampling according to a

[1] https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer
[2] http://www.gurobi.com

variety of footprint shapes and sizes and domains with holes are included. These libraries are the first step before the mixed-integer linear program is populated and solved. The repository includes additional Python and C++ routines for visualizing solutions.

4.3.2 Web-Based Collection Scheduler

Pyomo instances for the deterministic and stochastic collection scheduling models discussed in Section 1 are also included in the internal GitLab repository at https://gitlab.sandia.gov/82634/ops. Additionally, there is a branch with a web application written in a combination of Python and JavaScript to provide a web user interface for calling collection scheduling solvers and displaying and comparing schedules.

4.3.3 Other

Additional software was developed over the course of the project. For more information please speak with Christopher G. Valicka.

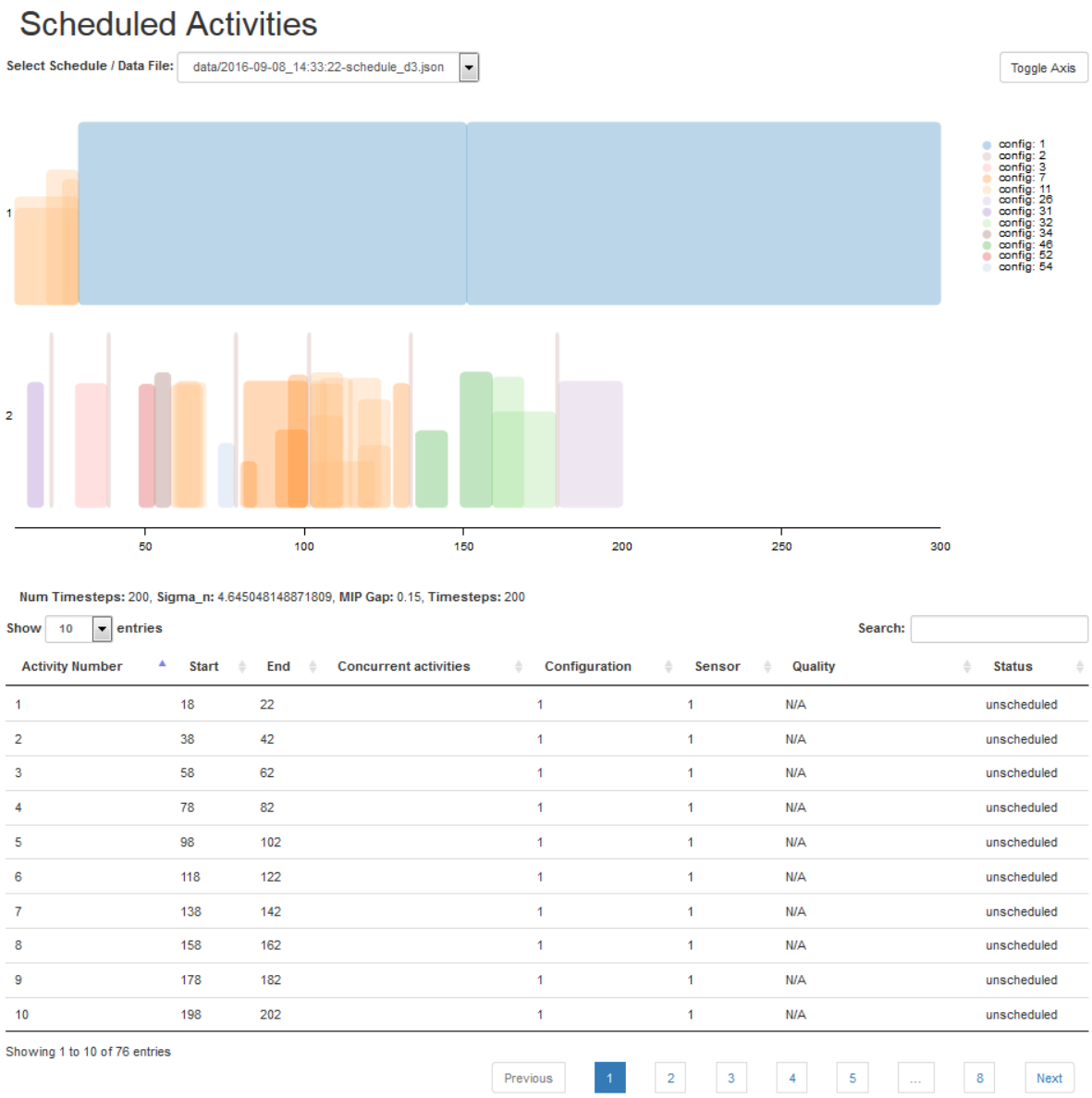


Figure 4.1. Main interface to web-based scheduler. Window displays each sensor's schedule and includes a table with information about scheduled and unscheduled collections.

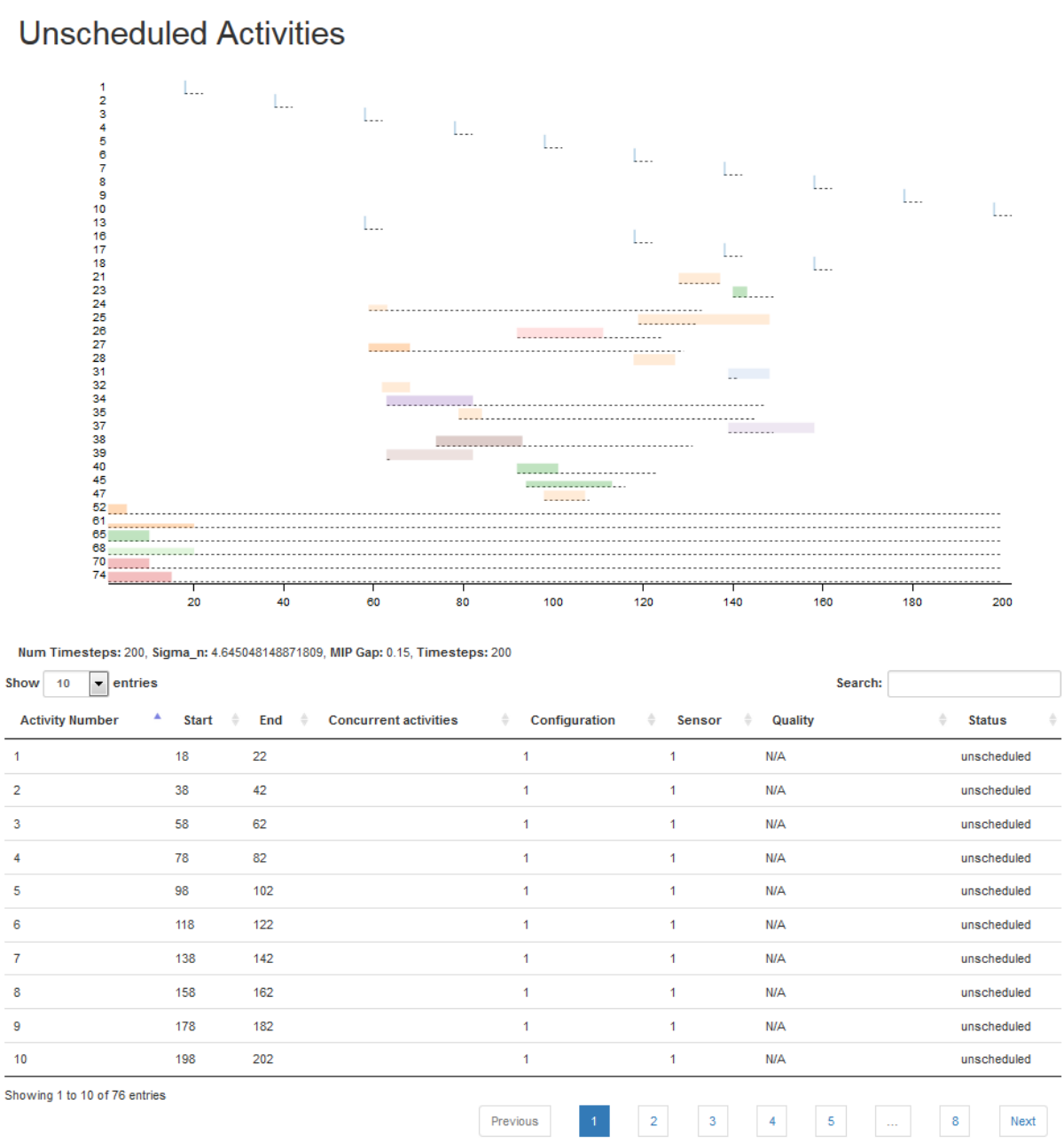


Figure 4.2. Unscheduled collections window displaying priority and duration information. Dotted lines denote when collection could have been scheduled and which sensors are feasible for accomplishing the collection.

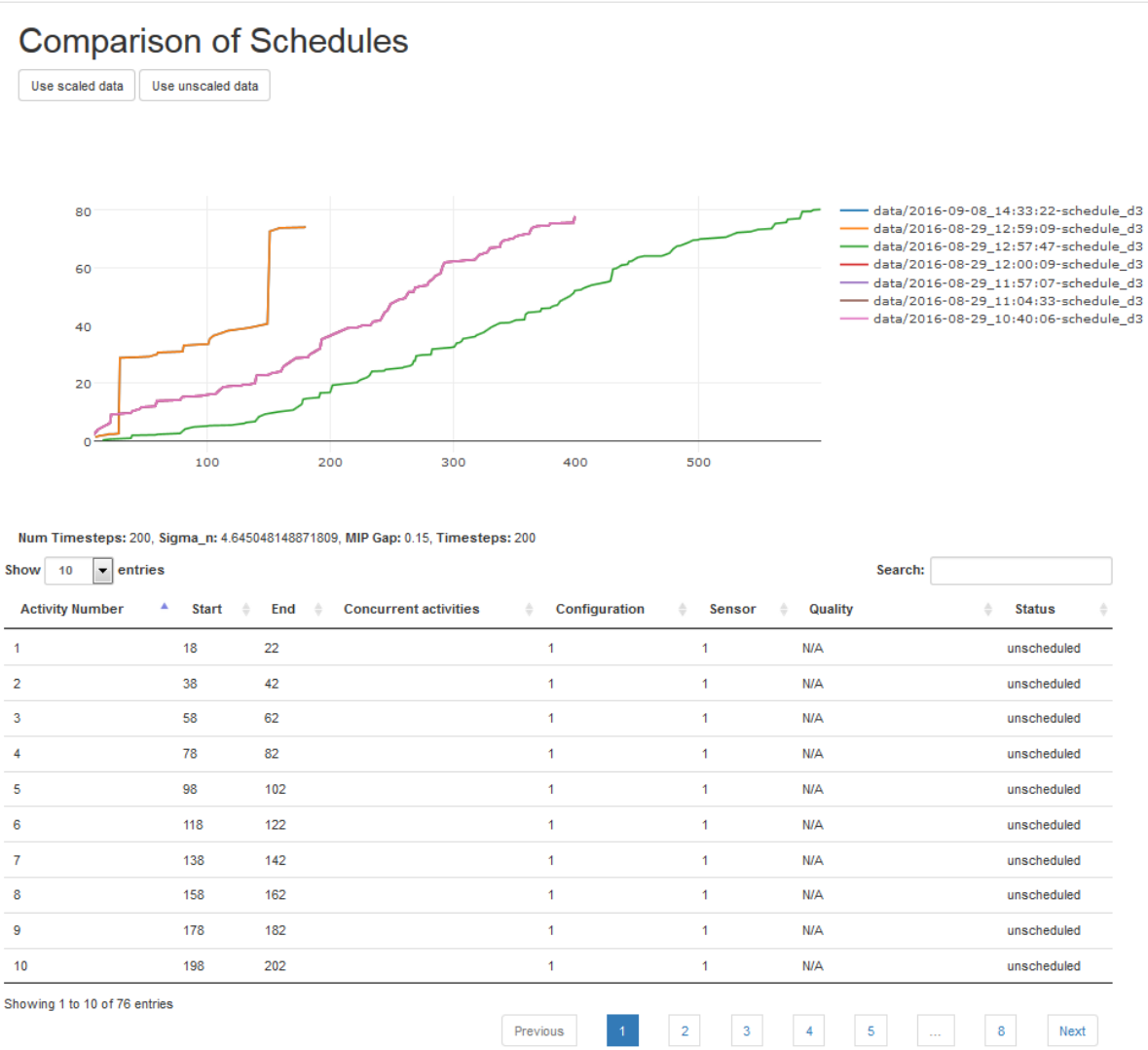


Figure 4.3. Window for displaying objective function value for different schedules. The aim is to allow solver developers or planners to quantitatively and qualitatively compare schedules through visual inspection.

Chapter 5

Collaboration with Texas A&M University

Over the course of the project, the Sandia team partnered closely with professors and students from Texas A & M University. Periodic teleconferences, visits to Albuquerque and College Station, and a project meeting at the Military Operations Research Society 84th Symposium played a formative role in the outcome of the project. The Texas A&M team brought expertise in UAV routing problems, mixed-integer linear optimization, and stochastic optimization. Included in Appendix A is a literature review comparing alternative mixed-integer programming formulations for the collection scheduling problem. Experimental results of heuristics developed for the collection scheduling problem are reported in [41]. The teams hope to conduct experiments with the stochastic formulation presented in [44] and develop heuristics for both presented stochastic collection scheduling models. Future work will also consist of developing heuristics for the model presented in Equation (1.32) and developing stochastic models for alternative sources of uncertainty and rewards associated with sophisticated scheduling strategies.

Chapter 6

Future Directions and Concluding Remarks

The tools and techniques explored and developed under this project have the potential to provide improvements to near- and long-term problems faced by sensor planners and operators. The new, rigorous models can serve as a tool to produce schedules and placements in operational time frames, study strategic constellation decisions, and be used as a means to compare different scheduling and placement heuristics. We see a number of topics as worthy of further exploration. Combining footprint placement with collection scheduling, combining sub-footprint placement with footprint placement, and all three, is a natural progression as the footprint and sub-footprint pieces were designed to be MIP's that could be integrated into the collection scheduling MIP. An additional complication when combining the footprint and sub-footprint problems is deciding which footprint to assign a sub-footprint to when it is in more than one. Combination of these different problems may naturally fit into constellations that aim to optimize a single collection window amongst multiple sensors or subdivide a collection window into portions requiring different sensor configurations or capabilities. Going forward, an emphasis on developing a community model of quality will be needed to help constrain or guide these MIP models toward sufficiently high fidelity and sufficiently low computational complexity. Factors that together define value of information or information gained from different Earth-observing or space surveillance network sensor modalities need to be defined relative to the other modalities available in each constellation.

We can solve problems requiring more footprint placements by using fewer samples, but the solutions tend to have significantly overlapping footprints if the epsilon sampling radius is more than about 10% of the footprint radius for circles, and the threshold is even less for squares and other shapes with sharp corners. Whether this is an issue is context dependent. For larger problems, a more geometric approach, as described by the alternatives in the introduction, may be more scalable. However, from a practical standpoint, most problems of interest in our context actually have only a few tens of footprint placements needed, and runtimes up to a few tens of minutes may be acceptable or reduced through software and solver tuning.

For the footprint and sub-footprint placement problems, we may consider speeding up the (approximate) solution to our mixed-integer programs (MIPs) by exploiting their problem structure, since solving them as standard black-box MIPs is slow. In particular, some form of the Lagrangian Heuristic of Daniels [7] may be helpful. One may consider iterative placement of footprints, and "pinning" a footprint's placement by an extremal point. Pinning may also be useful for the problem variation where the ROI is a set of discrete points. In certain circumstances one may show that for any solution with a footprint not pinned, there is an equivalent-quality solution with the

footprint pinned. One can show this is true for restricting placement points to the ROI convex hull for circular footprints, but not for many non-circular shapes. There may be other heuristics for defining an initial feasible placement, and for discarding branches in the branch and bound.

Related but separate is the idea of including constraints according to sensor deficiencies that are localized to certain regions within footprints or sub-footprints. Adding these constraints would be straightforward and could be very useful in capturing planners and operators sensor expertise to provide better mosaics and sub-footprint placements. For example, if a set of sensor pixels are bad, we would like to shift the chip layout so that sub-footprint placements would not include these bad pixels. Further extending the model to incorporate the uncertainty associated with geolocation accuracy could make use of stochastic mixed-integer programming modeling and solution capabilities of Pyomo.

For the collection scheduling models, further investigation of the model in Equation (1.32) should be pursued. It perhaps represents a natural way to combine the footprint and sub-footprint placement optimization solutions with the collection scheduling problem. As a more complex model, it will likely require additional runtime to generate optimal or near-optimal schedules. Efficient heuristics and clear comparisons of schedules produced using the model in Equation (1.6) will be key to operationalize this model. Experimentation with the *ad hoc* model presented in [44] will require generation of suitable benchmark datasets to capture the periodicity and variability of *ad hoc* collections. Experiments and investigations of alternative constellations should be pursued with the presented models.

Finally, there is a strong desire amongst today's single sensor and sensor constellation planners to effectively collect on ephemeral or uncertain targets. Modifications, re-purposing, or extension of the stochastic models developed under this LDRD will be key to achieving sensor and constellation-level increases in value of information gained from sensor schedules. These same stochastic models may also be well-suited for achieving complex scheduling strategies. For example, strategies to achieve synthetic persistence, for collecting patterns of life information, or for scheduling complex tipping and cueing strategies over one or more targets may be possible with these models.

References

[1] Ahmed Abdelkader, Scott A. Mitchell, and Mohamed S. Ebeida. Steiner point reduction in planar Delaunay meshes. In *Symposium on Computational Geometry, Young Researchers Forum*, 2014. available from `http://www.computational-geometry.org/YRF/cgyrf2014.pdf` and `http://www.cs.sandia.gov/~samitch/bibliography_2007.html#steiner-reduction`.

[2] Philippe Baptiste and Ruslan Sadykov. Time-indexed formulations for scheduling chains on a single machine: An application to airborne radars. *European Journal of Operational Research*, 203(2):476 – 483, 2010.

[3] Laura Barbulescu, Jean-Paul Watson, L. Darrell Whitley, and Adele E. Howe. Scheduling space–ground communications for the air force satellite control network. *Journal of Scheduling*, 7(1):7–34, 2004.

[4] J. R. Birge and F. V. Louveaux. *Introduction to Stochastic Programming*. Springer, New York, 1997.

[5] D-S. Chen, R.G. Batson, and Y. Dang. *Applied Integer Programming*. Wiley, 2010.

[6] IBM ILOG CPLEX. V12. 1: User's manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009.

[7] Karen Daniels, Arti Mathur, and Roger Grinde. A combinatorial maximum cover approach to 2D translational geometric covering. In *Canadian Conference on Computational Geometry (CCCG)*, pages 2–5, 2003.

[8] Ajay Deshpande, Taejung Kim, Erik D Demaine, and Sanjay E Sarma. A pseudopolynomial time $O(\log n)$-approximation algorithm for art gallery problems. In *Workshop on Algorithms and Data Structures*, pages 163–174. Springer, 2007.

[9] Qiu Dishan, He Chuan, Liu Jin, and Ma Manhao. A dynamic scheduling method of earth-observing satellites by employing rolling horizon strategy. *The Scientific World Journal*, 2013, 2013.

[10] Mohamed S. Ebeida, Muhammad A. Awad, Xiaoyin Ge, Ahmed H. Mahmoud, Scott A. Mitchell, Patrick M. Knupp, and Li-Yi Wei. Improving spatial coverage while preserving the blue noise of point sets. *Computer-Aided Design*, 46(0):25 – 36, 2014. 2013 SIAM Conference on Geometric and Physical Modeling, SIAM GD/SPM13. Reposting on author website with permission from Elsevier as U.S. government funded work: `http://www.cs.sandia.gov/~samitch/bibliography_2007.html#opt-beta`.

[11] Mohamed S. Ebeida, Ahmed H. Mahmoud, Muhammad A. Awad, Mohammed A. Mohammed, Scott A. Mitchell, Alexander Rand, and John D. Owens. Sifted disks. *Computer Graphics Forum, Proc. Eurographics*, 32(2pt4):509–518, 2013.

[12] Mohamed S. Ebeida, Scott A. Mitchell, Anjul Patney, Andrew A. Davidson, and John D. Owens. A simple algorithm for maximal Poisson-disk sampling in high dimensions. *Computer Graphics Forum, Proc. Eurographics*, 31(2):785–794, 2012.

[13] Mohamed S. Ebeida, Ahmad A. Rushdi, Muhammad A. Awad, Ahmed H. Mahmoud, Dong-Ming Yan, Shawn A. English, John D. Owens, Chandrajit L. Bajaj, and Scott A. Mitchell. Disk density tuning of a maximal random packing. *Computer Graphics Forum*, 35(5):–, 2016.

[14] Forecast.io. Forecast.io. Online; accessed 4-September-2015, 2015.

[15] Jeremy Frank, Ari Jónsson, Robert Morris, and David E. Smith. Planning and scheduling for fleets of earth observing satellites. In *in: Proceedings of Sixth Int. Symp. on Artificial Intelligence, Robotics, Automation & Space*, 2001.

[16] Al Globus, James Crawford, Jason Lohn, and Anna Pryor. A comparison of techniques for scheduling earth observing satellites. In *AAAI*, pages 836–843, 2004.

[17] Sariel Har-Peled. *Geometric approximation algorithms*, volume 173. Amer. Math. Soc., 2008.

[18] Sariel Har-Peled. Covering a simple polygon with circles. `http://cstheory.stackexchange.com/questions/8799/covering-a-simple-poly`, November 2011.

[19] William E Hart, Carl Laird, Jean-Paul Watson, and David L Woodruff. *Pyomo–optimization modeling in python*, volume 67. Springer Science & Business Media, 2012.

[20] William E Hart, Jean-Paul Watson, and David L Woodruff. Pyomo: Modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011.

[21] Tyler A Hobson and I Clarkson. Sensor-scheduling simulation of disparate sensors for space situational awareness. In *2011 Advanced Maui Optical and Space Surveillance Technologies Conference*, 2011.

[22] Gurobi Optimization Inc. Gurobi optimizer reference manual, 2015.

[23] Dae-Sung Jang and Han-Lim Choi. Discretization of planar geometric cover problems. *CoRR*, abs/1411.6810, 2014.

[24] Giorgos D. Kazazakis and Antonis A Argyros. Fast positioning of limited-visibility guards for the inspection of 2d workspaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002)*, volume 3, pages 2843 – 2848, Lausanne, Switzerland, October 2002. IEEE.

[25] Jun Li, Jun Li, Ning Jing, Weidong Hu, and Hao Chen. A satellite schedulability prediction algorithm for {EO} {SPS}. *Chinese Journal of Aeronautics*, 26(3):705 – 716, 2013.

[26] D. Y. Liao and Y. T. Yang. Imaging order scheduling of an earth observation satellite. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(5):794–802, Sept 2007.

[27] W. Mak, D.P. Morton, and R.K. Wood. Monte carlo bounding techniques for determining solution quality in stochastic programs. *Annals of Operations Research*, 24(1–2):47–56, 1999.

[28] Kassandra M Merritt. Coverage of continuous regions in Euclidean space using homogeneous resources with application to the allocation of the phased array radar systems. Master's thesis, Air Force Institute of Technology, 2011.

[29] Nabil H. Mustafa, Rajiv Raman, and Saurabh Ray. QPTAS for geometric set-cover problems via optimal separators. *Computing Research Repository (CoRR)*, abs/1403.0835, 2014.

[30] Cristina Neacsu and Karen Daniels. Translational covering of closed planar cubic b-spline curves. In *Computer Graphics Forum*, volume 25, pages 743–757. Wiley Online Library, 2006.

[31] L. Ntaimo. Disjunctive decomposition for two-stage stochastic mixed-binary programs with random recourse. *Operations Research*, 58:229–243, 2010.

[32] L. Ntaimo. Fenchel decomposition for stochastic mixed-integer programming. *Journal of Global Optimization*, 55:141–163, 2013.

[33] L. Ntaimo, J.A. Gallego Arrubla, C. Stripling, J. Young, and T. Spencer. A stochastic programming standard response model for wildfire initial attack planning. *Canadian Journal of Forest Research*, 42(6):987–1001, 2012.

[34] L. Ntaimo, J.A. Gallego Arrubla, C. Stripling, J. Young, and T. Spencer. A simulation and stochastic integer programming approach to wildfire initial attack planning. *Forest Science*, 59(1):105–117, 2013.

[35] Joseph C Pemberton and Flavius Galiber. A constraint-based approach to satellite scheduling. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 57:101–114, 2001.

[36] G. Peng, L. Wen, Y. Feng, B. Baocun, and Y. Jing. Simulated annealing algorithm for eos scheduling problem with task merging. In *Modelling, Identification and Control (ICMIC), Proceedings of 2011 International Conference on*, pages 547–552, June 2011.

[37] Artur Pessoa, Eduardo Uchoa, Marcus Poggi de Aragão, and Rosiane Rodrigues. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation*, 2(3):259–290, 2010.

[38] R.T. Rockafellar and R. J-B. Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, pages 119–147, 1991.

[39] A. Ruszczyn'ski and A. Shapiro, editors. *Stochastic Programming*, volume 10 of *Handbooks in Operations Research and Management Science*. Elsevier, 2003.

[40] Y. G. Stoyan and V. M. Patsuk. Covering a compact polygonal set by identical circles. *Computational Optimization and Applications*, 46(1):75–92, 2010.

[41] Kaarthik Sundar, Jianglei Qin, Sivakumar Rathinam, Lewis Ntaimo, Swaroop Darbha, and Christopher Valicka. Algorithms for a satellite constellation scheduling problem. In *2016 International Conference on Automation Science and Engineering*. IEEE, 2016.

[42] Kaarthik Sundar and Sivakumar. Rathinam. route planning algorithms for unmanned aerial vehicles with refueling constraints. In *2012 American Control Conference (ACC)*, pages 3266–3271. IEEE, 2012.

[43] Christopher G. Valicka, Deanna Garcia, Andrea Staid, Jean-Paul Watson, Gabriel Hackebeil, Sivakumar Rathinam, and Lewis Ntaimo. Models for optimal constellation scheduling under weather uncertainty. Manuscript submitted for publication, 2016.

[44] Christopher G. Valicka, Deanna Garcia, Andrea Staid, Jean-Paul Watson, Mark D. Rintoul, Gabriel Hackebeil, and Lewis Ntaimo. Sensor network scheduling under uncertainty: Models and benefits. In *2016 Advanced Maui Optical and Space Surveillance Technologies Conference*, 2016.

[45] Christopher G. Valicka, Mark D. Rintoul, William E. Hart, Scott A. Mitchell, Simon X. Zou, and Stephen Rowe. Mixed-integer formulations for constellation scheduling. In *2015 Advanced Maui Optical and Space Surveillance Technologies Conference*, 2015.

[46] JM Van den Akker, Cor AJ Hurkens, and Martin WP Savelsbergh. Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12(2):111–124, 2000.

[47] H. Wang, M. Xu, R. Wang, and Y. Li. Scheduling earth observing satellites with hybrid ant colony optimization algorithm. In *Artificial Intelligence and Computational Intelligence, 2009. AICI '09. International Conference on*, volume 2, pages 245–249, Nov 2009.

[48] Jianjiang Wang, Erik Demeulemeester, and Dishan Qiu. A pure proactive scheduling algorithm for multiple earth observation satellites under uncertainties of clouds. *Computers & Operations Research*, 74:1 – 13, 2016.

[49] J.-P. Watson, D.L. Woodruff, and W. Hart. Pysp: Modeling and solving stochastic programs in python. *Mathematical Programming Computation*, 4(2), 2012.

[50] J.P. Watson and D.L. Woodruff. Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Computational Management Science*, 8(4), 2011.

[51] Michael Weinreb, Michael Jamieson, Nancy Fulton, Yen Chen, Joy Xie Johnson, James Bremer, Carl Smith, and Jeanette Baucom. Operational calibration of geostationary operational environmental satellite-8 and-9 imagers and sounders. *Applied optics*, 36(27):6895–6904, 1997.

[52] William J Wolfe and Stephen E Sorensen. Three scheduling algorithms applied to the earth observing systems domain. *Management Science*, 46(1):148–166, 2000.

[53] Laurence A. Wolsey. *Integer Programming*. Wiley, 1998.

[54] Fatos Xhafa, Junzi Sun, Admir Barolli, Alexander Biberaj, and Leonard Barolli. Genetic algorithms for satellite scheduling problems. *Mob. Inf. Syst.*, 8(4):351–377, Oct 2012.

[55] Stephen L Zolnay. Earth coverage (”footprint”) of a satellite-borne antenna. Technical report, DTIC Document, 1971.

Appendix A

Texas A&M Literature Review and Comparison of Alternative Formulations

Formulations for the constellation scheduling problem

Abstract

This write-up defines the constellation scheduling problem, discusses various ways for formulating the constraints with relevant references.

Index Terms

constellation; scheduling; mixed-integer programming

I. Problem definition

The constellation scheduling problem aims to schedule a set of monitoring tasks on a collection of mobile sensors. The scheduling is a look-ahead type as in the case of certain power system applications. The look-ahead time period can range from a few hours to a day. The performance of the mobile sensors are evaluated using the tasks at hand. From here on, we refer to the tasks as *activities*. We assume that there are multiple sensors and each sensor is capable of handling a specific activity and every activity can be performed by at least one sensor. In the scheduling literature, this is equivalent to having parallel machines. Each activity is associated with a start time, a deadline, a duration, a priority, an hierarchical importance or category (mandatory, essential or desired) and a minimum required quality (this refers to the quality of the observation made by the sensor; this is often impacted by time of the day, weather etc.). Also, each activity has a utility which is a function of the time the activity starts, the sensor that performs the activity, and the activity itself. The goal of the problem is to schedule all the activities on the available sensors so that the sum of the utilities corresponding to the activities performed on the machines is maximized while satisfying all the scheduling restrictions of the activities. The constraints associated with all the activities are as follows:

- There are precedence relations between activities which can be represented by an acyclic directed graph.
- Each activity belongs to one of the the three categories: 1. mandatory, 2. essential, or 3. desired. Mandatory activities have to be performed by some machine within its time window, essential activities are high priority activities and the rest of the activities are desired activities.
- The activities are non-preemptive *i.e.,* the activities cannot be split into smaller parts. Once a sensor starts to perform an activity, it has to complete the activity before taking up the next activity.

II. Overview of the Approach

The constellation scheduling problem belongs to the class of multiple machine, multiple job scheduling problems, and even simple cases of this problem are strongly NP-Hard[14]. There are generally two ways of formulating these problems. One way is to formulate the problem as a mixed integer linear program where the completion time or the start time of an activity is a continuous time variable. Another way is to discretize time and formulate an integer program (also referred to as time-indexed formulation). In this project, we will chose the later for the following reasons: 1) Computing the utility associated with performing an activity on a machine is a non-linear function of time, and as a result, the utility data is easier to represent if the time is discretized; 2) The formulations with discrete time variables for scheduling problems are known to produce better bounds for estimating the optimum [1]. Even though discretizing time leads to formulations with large number of variables, we can adopt one of the following approaches to improve the computational performance.

- Time-indexed formulations can be posed as network flow type problems with side constraints and are more amenable to Lagrangian dual based heuristics. Recently, these heuristics have produced near-optimal solutions for similar scheduling problems in the literature [9, 17, 2].
- A time-indexed scheduling problem can be reformulated so that the resulting transformation is amenable to Dantzig-Wolfe decomposition methods which can handle large scale relaxations of the problem. Numerical results in [3, 1] have shown that this approach has been able to solve large time-indexed parallel machine scheduling problems in different applications.
- The time-indexed formulation can be further strengthened by tracking the sequence of activities performed on a machine as a function of time. Recently, branch, cut and price methods have been developed for these generalizations with computational results showing superior performance [14].

In the next section, we provide an overview of the above solution approaches used in solving a time-indexed formulation for scheduling problems.

III. Solution approaches

An important advantage of using a time-indexed formulation is that, for many variants of the scheduling problem, it can easily accommodate a diverse set of constraints and objective functions, and is a natural way of posing the problem. Furthermore, the linear programming (LP) relaxation of time-indexed formulations is known to provide good lower bounds [1]. However, the main disadvantage of the time-indexed formulation is its size. As a result, advanced computational techniques are required to handle instances with a large number of jobs as the resulting formulations involve a large number of variables and constraints. In the next subsection, we discuss one of these techniques based on Dantzig-Wolfe decomposition.

A. Dantzig-Wolfe decomposition

Dantzig-Wolfe decomposition is a way of reformulating the problem to reduce the number of constraints at the expense of increasing the number of variables. But, the huge number of variables will not pose a problem as they can be handled implicitly using column generation techniques. Dantig-Wolfe decomposition can be applied to any linear programming relaxation of the scheduling problem with the following structure:

$$\begin{aligned} \min \quad & cx \\ Ax &\leq b, \\ x &\in P, \end{aligned}$$

where P is a polytope. The theory extends for the case where P is unbounded. We will use the following theorem for convex polyhedra to develop a reformulation for the above linear program:

Theorem 1. *Minkowski-Weyl's theorem: For a subset P of R^d, the following statements are equivalent:*

1) *P is a polytope,* i.e., *$P = \{x, Ax \leq b\}$;*
2) *There are finite real vectors $v_1, \ldots, v_n \in R^d$ such that $P = conv(v_1, \ldots, v_n)$.*

The first representation of a polytope is known as the *H-representation* (halfspace) and the second is referred to as the *V-representation* (vertex). Dantzig-Wolfe decomposition expresses the polytope P in the V-representation; each vector $x \in P$ can be represented as

$$x = \sum_{k=1}^{n} \lambda_k x^k, \text{ with } \sum \lambda_k = 1, \quad \lambda_k \geq 0, \quad (k = 1, \ldots, N).$$

Substituting these equations in the original linear programming relaxation leads to what is knows as the Dantzig-Wolfe master problem :

$$\begin{aligned} \min \quad & \sum_{k=1}^{n} (cx^k)\lambda_k \\ & \sum_{k=1}^{n} (Ax^k)\lambda_k \leq b, \\ & \sum_{k=1}^{n} \lambda_k = 1, \\ & \lambda_k \geq 0. \end{aligned}$$

Observe that the reformulation has fewer constraints and a large number of variables compared to the original problem. We can circumvent this difficulty by using column generation techniques to handle these variables implicitly rather than explicitly. The overall principle of the column generation is as follows: (i) start with a subset of variables (while fixing other

variables implicitly to zero), (ii) solve the restricted master problem and check if the current objective may be improved by including some variables that are currently implicitly fixed to zero, (iii) if any such variables exist, add them to master problem and reoptimize. The identification of variables that can improve the current solution objective is done by solving what is called a pricing problem. The pricing problem utilizes the optimal dual solution to the restricted master problem to identify the improving variables. This was the basic approach used for solving time-indexed formulations of the scheduling problem in [3, 1]. The above principle can be used together with the branch-and-bound framework to solve mixed-integer programs to optimality. This technique of solving mixed-integer linear programs is called branch-and-price.

B. Lagrangian relaxation

Lagrangian relaxation is one of the techniques that can be applied to problems where the constraints can be divided into two sets [6] (i) "easy" constraints that are relatively straightforward to handle, and (ii) "hard" constraints that make the problem very difficult to solve. The main idea is to relax the problem by removing the "hard" constraints and penalizing them in the objective function whenever these constraints are violated. This technique has been successfully employed to compute high quality solutions for single and multiple machine scheduling problems (see [9, 17, 2]).

To further illustrate this approach, consider the following integer linear program for a scheduling problem:

$$Z_{IP} := \min \quad cx$$

$$Ax \leq b$$

$$Dx \leq d$$

$$x \text{ integer}$$

Let $X := \{x \text{ integer } : Dx \leq d\}$ and assume that optimizing over the set X can be done relatively easily and $Ax \leq b$ are the "hard" constraints that make problem intractable. Let $\lambda \geq 0$ denote the vector of penalty or dual variables corresponding to the constraints in $Ax \leq b$. Now consider the relaxed problem

$$Z(\lambda) := \min \quad cx + \lambda(b - Ax)$$

$$x \in X.$$

The constraints are partitioned into sets so that the above relaxed problem can be solved efficiently. For example, in [9], the relaxation for the scheduling problem requires one to find a shortest path on an interval graph which can be done efficiently. In [2], the relaxation for

the scheduling problem requires one to find an optimal stable set on an interval graph which again is relatively easier to solve.

The "weak duality" theorem for convex optimization states that $Z(\lambda)$ provides a lower bound on Z_{IP}. Therefore, the best lower bound for the optimal cost can be obtained by solving the following Lagrangian dual: $Z_D := \max_{\lambda \geq 0} Z(\lambda)$. For integer programs, we only have $Z_D \leq Z_{IP}$. Heuristics are often employed to convert this lower bound to a feasible solution of very high quality. Therefore, the quality of the feasible solutions depend of how efficiently we can compute Z_D. It is known that the objective in Z_D is concave, piecewise linear, and non-differentiable. Hence, the classical approach of finding a maximum using gradient methods will not work. In the literature, one resorts to subgradient optimization techniques to solve the Lagrangian dual problem. This was precisely the approach taken in [2] to compute high quality feasible solutions for machine scheduling problems.

IV. Stochastic versions of the constellation scheduling problem

Stochastic mixed-integer programming (SMIP) allows for modeling MIPs with data uncertainties. Therefore, the constellation scheduling problem can be formulated using SMIP. In the constellation problem, the assignment of activities to sensors must take into account the *weather* conditions at the location to be observed. Therefore, weather is a source of data uncertainty and weather forecast data can be used using the SMIP framework in order to obtain schedules that hedge against poor observation quality due to possible adverse weather conditions. Another source of data uncertainty is the effect of "category 4 activities" on the sensor schedules. These activities have start time windows that are time-varying due to the stochastic nature of the phenomenon being to be observed (e.g. wildfires or flooding). Therefore, the occurrence of the category 4 activity is probabilistic and they have to be scheduled so that there is minimal disruption to the schedule in terms of what activities are displaced.

We will investigate including a combination of the above uncertainties in the problem formulated using stochastic programming with recourse. These uncertainties will be modeled as random variables using probability distributions. Hence, special algorithms will be designed for the resulting stochastic programming formulations with recourse. In practice, empirical probability distributions can be constructed based on historical data or prediction models and used in the formulations. There is very limited work on the stochastic version of the constellation scheduling problem. So next we give a brief review of the SMIP approach.

SMIP with Recourse

The constellation scheduling problem can be formulated using SMIP as a two-stage stochastic mixed (0,1) program with recourse. In the SMIP paradigm, a decision must be made here-and-now in the first-stage before future uncertainty (second-stage) is revealed. Future uncertainty is modeled using probability distributions of the random variables in the problem. A *recourse* decision is then made in second-stage after the uncertainty is resolved. The recourse action

allows to hedge against possible future outcomes (scenarios) so that the first-stage decision is optimized with this information taken into account. Mathematically, generic two-stage stochastic mixed (0,1) programming formulation can be given as follows:

$$\begin{aligned} \text{SIP2: } & \min c^\top x + \mathbb{E}[f(x, \tilde{\omega})] \\ & \text{s.t. } Ax \geq b \\ & x \in X. \end{aligned} \tag{1}$$

In problem SIP2, x denotes the first-stage decision vector, $c \in \mathbb{R}^{n_1}$ is the first-stage cost vector, $b \in \mathbb{R}^{m_1}$ is the first-stage right-hand side, $f(x, \tilde{\omega})$ is the recourse function with $\tilde{\omega}$ being a multivariate random variable, and $\mathbb{E}$ denotes the mathematical expectation operator satisfying $\mathbb{E}[|f(x, \omega\tilde{\omega})|] < \infty$ for all $x \in \{Ax \geq b, x \in X\}$. $A \in \mathbb{R}^{m_1 \times n_1}$ is the first-stage constraint matrix, and X defines binary restrictions on some components of x. The decision vector x includes all scheduling decisions that have to be made here-and-now or in the nearest future. The constraints $Ax \geq b$ define the scheduling constraints corresponding to the here-and-now decisions.

If we assume that the underlying probability distribution of $\tilde{\omega}$ is discrete with a finite number of realizations (scenarios) and corresponding probabilities $p_\omega, \omega \in$, then for a given scenario ω, the recourse function $f(x, \omega)$ is given by the following second-stage mixed (0,1) program:

$$\begin{aligned} f(x, \omega) = \min & q_\omega^\top y_\omega \\ & \text{s.t. } W_\omega y_\omega \geq h_\omega - T_\omega x \\ & y_\omega \in Y. \end{aligned} \tag{2}$$

In formulation (2), $y_\omega \in \mathbb{R}^{n_2}$ is the recourse decision variable vector, $q_\omega \in \mathbb{R}^{n_2}$ is the recourse cost vector, $W_\omega \in \mathbb{R}^{m_2 \times n_2}$ is the recourse matrix, $T_\omega \in \mathbb{R}^{m_2 \times n_1}$ is the technology matrix, and $h_\omega \in \mathbb{R}^{m_2}$ is the right-hand side. The set Y defines binary restrictions on some components of y_ω.

Decomposition Approaches

Solving instances of SP2 is generally challenging and we explore decomposition approaches. Traditionally, decomposition methods fall under one of two categories: *stage-wise* decomposition and *scenario-wise* decomposition. Stage-wise decomposition strategies are usually based on Benders' decomposition [4] or L-shaped decomposition [18]. Scenario-wise decomposition strategies use variable splitting procedures on the first-stage decision variables to create *nonanticipativity* constraints, and then relax them using Lagrangian relaxation. In both approaches, solving scenario MIP problems several times is a daunting task and usually the LP relaxation of the problems are solving using a cutting plane, branch-and-bound, or branch-and-cut scheme.

Fenchel Decomposition: This is a relatively recent approach [13] and uses Fenchel cutting planes from IP in a stage-wise decomposition setting. Considering x as a first-stage decision

variable and y as the second-stage decision variable, two forms of cuts called *Fenchel decomposition* (FD) *cuts* are used following the Benders decomposition setting. This approach has shown good computation performance for some standard test instances. Another approach similar to FD is the disjunctive decomposition (D2) method [12]. Unlike the FD method, this D2 method uses disjunctive cutting planes.

Scenario or dual decomposition for SP2 was devised by [5]. In this method, the Lagrangian multipliers associated with the relaxed nonanticipativity constraints are updated from one iteration to the next using the subgradient optimization bundle method [8]. Since the dual decomposition algorithm relies on the Lagrangian dual, it may not be possible to find the (integer) optimal solution to SP2 because of the existence of the duality gap for the Lagrangian dual of an IP [11]. Therefore, a BAB procedure has to be implemented to achieve finite convergence to the integer optimal [5]. Another study of the dual decomposition algorithm can be found in [10].

Another scenario decomposition approach is the progressive hedging algorithm (PHA) [15] for solving stochastic linear programs. The PHA has also been applied to SP2 but the issue is relatively slow convergence to an optimal integer solution [7]. The convergence of PHA subproblems to a nonanticipative first-stage solution is not based on Lagrangian duality, but the form of the subproblem is similar to the dual decomposition subproblem. It differs by including a regularization quadratic term in the objective to penalize moving too far from an incumbent nonanticipative solution.

To enable scalability of the decomposition methods for SP2, sampling approaches such as the sample average approximation (SAA) scheme [16] should be considered. This allows for solving SP2 based on sampling a limited number of scenarios to reduce the size of the problem and enable relatively faster solutions. This can be accomplished in conjunction with exploiting the problem structure. Finally, to enable timely decisions imposed by the problem, parallel/distributed implementations of the algorithms can also be considered.

In the next section, the constellation scheduling problem is posed using a time-indexed formulation. This formulation can be used as a basis for adopting any of the approaches mentioned above.

V. Notations and formulations

This section introduces the nomenclature that is used throughout the rest of the article. We will then briefly present a few formulations for the constellation scheduling problem defined in Sec. I.

Nomenclature

T scheduling time horizon;

t time period index, $t = 1, \ldots, T$;

J set of activities indexed by j;

I set of mobile sensors indexed by i;
K_1 set of mandatory activities;
K_2 set of essential activities;
K_3 set of desired activities;
K $K_1 \cup K_2 \cup K_3$, indexed by k;
G precedence graph (directed acyclic);
$pred_j$ set of predecessor activities for j;
$succ_j$ set of successor activities for j;
rel_j release times for activity j;
due_j due time for activity j;
δ_j deadline for activity j;
t_j $[rel_j, \delta_j]$ - time window for activity j;
p_j processing time for activity j;

A. Time-indexed formulation

Let T_S (respectively T_E) be a lower (respectively upper) bound on the time the execution of any activity j can be completed; the values can be computed as $T_S = \min\{rel_j + p_j \,|\, j \in J\}$ and $T_E = \max\{\delta_j \,|\, j \in J\}$. This formulation uses binary decision variables $x^i_{jt} \in \{0, 1\}$ with $T_S \leq t \leq T_E$, where $x^i_{jt} = 1$ if activity j is completed exactly at time t on sensor i and 0 otherwise. We define the tardiness of an activity as $T_{jt} = (t - due_j)^+$. As mentioned in Sec. I, each activity j is associated with a utility β^i_{jt}, where i is the sensor on which the activity is performed. Using the notations and variables defined above, we can consider a number of objectives (we will present two commonly used in constellation scheduling) as follows:

OBJECTIVE: We can consider a utility maximization objective or a tardiness minimization objective or a combination of both. First let us consider the utility maximization objective. The utility associated with an activity is defined to depend on the start time of a activity. This assumption can also be reversed to make the utility a function of the completion time of the activity. Hence, the objective would be of the form

$$\max \quad \sum_{i,j} \sum_{t=rel_j+p_j}^{\delta_j} \beta^i_{jt} x^i_{jt} \tag{3}$$

The start time of any activity j can be obtained from x^i_{jt} as $(t - p_j)^+ x^i_{jt}$. The second objective that we can formulate is a tardiness minimization objective as

$$\min \quad \sum_{i.j} \sum_{t=rel_j+p_j}^{\delta_j} w_j T_{jt} x^i_{jt} \tag{4}$$

We could also combine the two objectives as

$$\max \quad \sum_{i,j} \sum_{t=rel_j+p_j}^{\delta_j} (\beta^i_{jt} - w_j T_{jt}) x^i_{jt} \tag{5}$$

CONSTRAINTS: Using the decision variables and nomenclature introduced above, we formulate the constraints of the problem as follows:

1) ACTIVITY OVERLAP: This set of constraints ensure that no two activities overlap in any sensor.

$$\sum_{j} \sum_{s=\underline{t}}^{\bar{t}} x_{js}^{i} \leq 1 \quad \forall i, T_S \leq t \leq T_E \tag{6}$$

In the above constraint $\underline{t} = \max\{t, rel_j + p_j\}$ and $\bar{t} = \max\{\delta_j, t + p_j - 1\}$. These constraints also ensure that the activities are being performed nonpreemptively.

2) CATEGORIES: These constraints enforce that the mandatory category activities have to be performed by some sensor.

$$\sum_{i} \sum_{t=rel_j+p_j}^{\delta_j} x_{jt}^{i} = 1 \quad \forall j \in K_1 \tag{7}$$

$$\sum_{i} \sum_{t=rel_j+p_j}^{\delta_j} x_{jt}^{i} \leq 1 \quad \forall j \in K \setminus K_1 \tag{8}$$

3) PRECEDENCE: The precedence constraints enforce the precedence constraints within the activities. We use the precedence graph G to enforce these constraints. Suppose (j_1, j_2) is an edge in the precedence graph where $j_1, j_2 \in J$, then we have

$$\sum_{i} \sum_{s=rel_{j_1}+p_{j_1}}^{\delta_{j_1}} x_{j_1 s}^{i} s \leq \sum_{i} \sum_{t=rel_{j_2}+p_{j_2}}^{\delta_{j_2}} x_{j_2 t}^{i} t - p_{j_2} \tag{9}$$

The above constraint enforces the precedence constraint globally over all the sensors. This can also be replaced with a local constraint for each sensor by considering a precedence graph for each sensor separately.

4) BINARY RESTRICTIONS:

$$x_{jt}^{i} \in \{0, 1\} \quad \forall i, j, rel_j + p_j \leq t \leq \delta_j \tag{10}$$

B. Alternative time-indexed formulation

An alternative formulation is obtained by replacing the precedence constraints in (9) by the following constraint for every edge (j_1, j_2) in the precedence graph G, for every t such that $\max\{rel_{j_1}, rel_{j_2} + p_{j_2}\} + p_{j_1} \leq t \leq \min\{\delta_{j_1}, \delta_{j_2} + p_{j_1}\}$:

$$\sum_{i} \left(\sum_{s=1}^{\delta_{j_1}} x_{j_1 s}^{i} + \sum_{s=rel_{j_2}+p_{j_2}}^{t-p_{j_2}} x_{j_2 s}^{i} \right) \leq 1 \tag{11}$$

It is known that the formulation in Sec. V-B is tighter than the formulation in Sec. V-A.

References

[1] J.M. van den Akker, C.A.J. Hurkens, and M.W.P. Savelsbergh. "Time-Indexed Formulations for Machine Scheduling Problems: Column Generation". In: *INFORMS Journal on Computing* 12.2 (2000), pp. 111–124.

[2] Pasquale Avella, Maurizio Boccia, and Bernardo D'Auria. "Near-Optimal Solutions of Large-Scale Single-Machine Scheduling Problems". In: *INFORMS J. on Computing* 17.2 (Apr. 2005), pp. 183–191.

[3] Philippe Baptiste and Ruslan Sadykov. "Time-indexed formulations for scheduling chains on a single machine: An application to airborne radars". In: *European Journal of Operational Research* 203.2 (2010), pp. 476 –483.

[4] J. Benders. "Partitioning procedures for solving mixed-variables programming problems". In: *Numerische Mathematik* 4 (1962), pp. 238–252.

[5] C. C. Caroe and R. Schultz. "Dual decomposition in stochastic integer programming". In: *Operations Research Letters* 24.1-2 (1999), 37–45.

[6] Marshall L Fisher. "An applications oriented guide to Lagrangian relaxation". In: *Interfaces* 15.2 (1985), pp. 10–21.

[7] Kjetil Haugen, Arne Løkketangen, and Daniel Woodruff. "Progressive hedging as a metaheuristic applied to stochastic lot-sizing". In: *Eurpoean Journal of Operational Research* 132 (2001), pp. 116–122.

[8] K. Kiwiel. "Proximity control in bundle methods for convex nondifferentiable minimization". In: *Mathematical Programming* 46 (1990), pp. 105–122.

[9] Youngho Lee and HanifD. Sherali. "Unrelated machine scheduling with time-window and machine downtime constraints: An application to a naval battle-group problem". English. In: *Annals of Operations Research* 50.1 (1994), pp. 339–365.

[10] F. V. Louveaux and R. Schultz. "Stochastic Integer Programming". In: *Stochastic Programming*. Ed. by A. Ruszczyǹski and A. Shapiro. Vol. 10. Handbooks in Operations Research and Management Science. Elsevier, 2003, pp. 213–266.

[11] G Nemhauser and L Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1999.

[12] L. Ntaimo. "Disjunctive Decomposition for Two-Stage Stochastic Mixed-Binary Programs with Random Recourse". In: *Operations Research* 58 ().

[13] L. Ntaimo. "Fenchel Decomposition for Stochastic Mixed-Integer Programming". In: *Journal of Global Optimization* 55 (2013), pp. 141–163.

[14] Artur Pessoa et al. "Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems". English. In: *Mathematical Programming Computation* 2.3-4 (2010), pp. 259–290.

[15] R. Rockafellar and R. Wets. "Scenarios and Policy Aggregation in Optimization under Uncertainty". In: *Mathematics of Operations Research* 16.1 (1991), pp. 119–147.

[16] A. Ruszczyński and A. Shapiro, eds. *Handbooks in Operations Research and Management Science: Stochastic Programming*. Vol. 10. Elsevier, 2003.

[17] Hanif D. Sherali, Youngho Lee, and Donald D. Boyer. "Scheduling target illuminators in naval battle-group anti-air warfare". In: *Naval Research Logistics (NRL)* 42.5 (1995), pp. 737–755.

[18] R. Van Slyke and R.-B. Wets. "L-shaped linear programs with application to optimal control and stochastic programming". In: *SIAM Journal on Applied Mathematics* 17 (1969), pp. 638–663.

DISTRIBUTION:

1	MS 1324	John T. Feddema, 1460
1	MS 0986	Brian N. Post, 2666
1	MS 0576	John H. Ganter, 5537
1	MS 1243	Cindi S. Reyes, 5523
1	MS 0980	David D. Cox, 5554
1	MS 0519	Kristina R. Czuchlewski, 5346
1	MS 0359	D. Chavez, LDRD Office, 1911
1	MS 0899	Technical Library, 9536 (electronic copy)


Sandia National Laboratories