A Decomposition Algorithm for Optimal Selection and Placement of Heterogeneous Sensors to Holistically Satisfy Mission

Michael Bynum, Forest Danford
Sandia National Laboratories
Georgia Stinchfield, Carl Laird
Carnegie Mellon University

Cody Karcher
California State University, Long Beach

ABSTRACT

Optimal selection, placement, and scheduling of terrestrial remote sensing systems is critical for effective utilization of valuable assets for space situational awareness missions. This problem is extremely challenging when selecting from a set of heterogeneous sensors within realistic budget under scenarios where the number of active targets exceeds the number of sensors. Scalable sensor placement algorithms are well established for the case where all sensors are static [1]; however, optimizing sensor placement is far more challenging when both stationary and dynamic sensors are candidates. Our exemplar considers both stationary and gimbaled sensors, which requires incorporating physical scheduling constraints (e.g., slew times) within the sensor placement problem to properly evaluate the merits of each type of sensor. The resulting problem is a large-scale mixed-integer linear program (MILP) that is intractable for full-scale problems of interest. We present a temporal decomposition algorithm for efficiently and optimally selecting both where to place sensors and the type of sensor(s) that should be selected at each location given a budget in order to satisfy mission criteria. The decomposition algorithm exploits problem structure induced by the temporal aspects of the scheduling constraints to solve a set of significantly smaller MILPs (in parallel) within a custom branch and bound (B&B) algorithm. This approach is broadly extensible based on the formulation chosen and the B&B algorithm is guaranteed to converge to the optimal solution. The decomposition algorithm can significantly improve computational performance. For one test problem, the decomposition algorithm reduced the optimality gap from 429.6% to 1.6% with a one-hour time limit.

1. NOTATION

Sets

$\mathbb{J}$	Set of jobs to be done
$\mathbb{S}$	Set of candidate sensors (location and technology)
$\mathbb{S}_s$	Set of static candidate sensors (location and technology)
$\mathbb{T}_s$	Set of candidate tasks for sensor s
$\mathbb{T}_{s,j}$	Set of candidate tasks for sensor s that can perform job j
$\mathbb{F}_s$	Set of tasks allowed to be the first task for sensor s
$\mathbb{L}_s$	Set of tasks allowed to be the last task for sensor s
$\mathbb{A}_s$	Set of feasible arcs (u,v) for sensor s where $u,v \in \mathbb{T}_s$
$\mathbb{C}_s$	Set of jobs that can be completed by static sensor s
$\mathbb{P}$	Set of time partitions for decomposition

Parameters

W_j	Weight or priority for job j
C_s	Cost of sensor s
B	Budget
H	Time horizon of interest

Variables

y_j	1 if job j is completed; 0 otherwise
$y_{s,j}$	1 if job j is completed by sensor s; 0 otherwise
$x_{s,t}$	1 if task t is selected for sensor s; 0 otherwise
$z_{s,u,v}$	1 if task u immediately precedes task v for sensor s; 0 otherwise
$f_{s,t}$	1 if task t is the first task for sensor s; 0 otherwise
$l_{s,t}$	1 if task t is the last task for sensor s; 0 otherwise
q_s	1 if sensor s is selected; 0 otherwise

2. INTRODUCTION

Optimal design of systems of sensors is crucial for effective utilization of valuable assets. Sensor system design involves both selection of sensor technologies and sensor placement. Optimization can reduce the number of sensors needed to achieve a particular objective or improve the utilization of a sensor system given a set budget. Given its importance, sensor system design and optimization has been studied across many different applications, including water contamination warning systems, gas detector placement for process safety, surveillance, and constellation design [1–5]. The utility of such optimization frameworks has even led to the development of general-purpose software for sensor placement optimization [6].

Mixed-integer programming (MIP) has proven to be an effective and scalable solution technique for many applications, even when considering false negatives [7, 8]. These methods typically decouple the physics from the optimization problem by pre-computing what each sensor can detect. For example, computational fluid dynamics (CFD) simulations were run to generate gas leak scenarios under varying weather conditions in [1]. These CFD simulations can be used to compute whether or not the concentration of the toxic gas reaches a high enough level at each candidate sensor location to be detected. The CFD simulations could be run in parallel, and the input to the optimization problem then becomes a list of scenarios each candidate location can detect. In this way, the CFD simulations do not need to be embedded within the MILP formulation, enabling the optimization problem to be solved at scale. With this methodology, both the types and locations of all sensors in the system can be simultaneously optimized. However, the majority of prior research has only considered static sensors where no operational decisions have to be made after placement.

In this paper, we consider the design of space situational awareness (SSA) remote sensing systems, including both static and gimbaled sensors. We consider scenarios with a set of jobs that may be completed by all or a subset of the sensor technologies considered. The gimbaled sensors must be scheduled during operations (if selected). Because the schedule of one gimbaled sensor depends on the types and locations selected for the other sensors, we cannot pre-compute the set of jobs that may be completed by each gimbaled sensor. To properly differentiate between the static and gimbaled sensors, we embed the scheduling problem inside of a mixed-integer linear program (MILP). The resulting problem is much larger than the corresponding problem with only static sensors and can be computationally challenging for some scenarios.

In order to solve the integrated sensor system design and scheduling problem, we propose a tailored decomposition algorithm that exploits the problem structure to accelerate the solution process. The remainder of this paper is organized as follows. In section 3, we present an MILP formulation to optimally design a remote sensing system for SSA given a budget, including both selection of sensor locations and selection of sensor technology at each location.

The MILP formulation considers both static and gimbaled sensors. In section 4, we detail the proposed decomposition algorithm. Section 5 contains computational results showing the advantages of the decomposition algorithm, and Section 6 summarizes the paper with conclusions.

3. PROBLEM FORMULATION

In this section, we present an MILP to optimally design a sensor system, considering both static and gimbaled sensors. We consider a representative time horizon, H. Let $\mathbb{J}$ be the set of jobs that need completed by at least one sensor within H. For design problems like the one considered here, it is important that $\mathbb{J}$ be representative of the set of jobs that may need to be done over any time horizon of equal length to H for the duration of the life of the sensors. If it is not possible to create the set $\mathbb{J}$ to be sufficiently representative, then multiple sets of jobs can be considered within a stochastic programming extension of the MILP presented here.

Each job $j \in \mathbb{J}$ must be completed within a specified time interval which is typically much shorter in length than H. Figure 1 shows an example of a job interval. Finally, each job has a corresponding priority or weight, W_j.

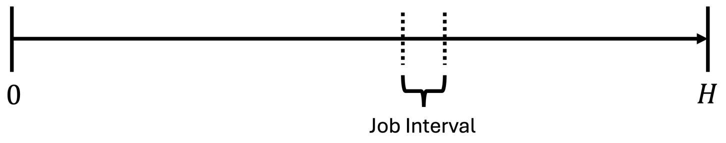

Fig. 1: Example of a job interval. The job must be completed within this interval.

Let $\mathbb{S}$ be the set of candidate sensors. Each $s \in \mathbb{S}$ represents both the sensor technology and the sensor location. Therefore, if one considers every sensor technology at every candidate location, the size of $\mathbb{S}$ is equal to the number of sensor technologies times the number of candidate locations. For the remainder of the paper, we use the term sensor interchangeably with sensor-location pair.

For each sensor, s, we generate a set of candidate tasks, $\mathbb{T}_s$, that the sensor may do if selected. Each task, $t \in \mathbb{T}_s$ is defined by a pointing direction and a task time. We represent the set of feasible sequences of tasks for each sensor using a directed acyclic graph (DAG) known as a feasibility DAG [9]. Each node in the DAG corresponds to a candidate task, $t \in \mathbb{T}_s$. There is an edge between two nodes u and v only if it is possible to complete task u and transition to task v before the start time of task v. The DAG is directed because the edges point from the task with the earlier time to the task with the later time. The use of a feasibility DAG drastically simplifies the scheduling portion of the optimization problem because the computations required to determine which edges should exist can be performed independently from the solution of the optimization problem. Thus, the scheduling portion of the problem becomes a graph traversal problem. To further reduce computational complexity, we remove any transitive edges in the graph. An edge from node u to node v is said to be transitive if there exists another path from u to v. This transitive reduction does not change the optimal objective value or the optimal selection of sensor technologies and locations.

Let $x_{s,t}$ be a binary variable that is one if task t is selected for sensor s and zero otherwise. Let $z_{s,u,v}$ be a binary variable that is one if task u immediately precedes task v for sensor s and zero otherwise. Let $f_{s,t}$ be a binary variable that is one if task t is the first task for sensor s and zero otherwise. Let $l_{s,t}$ be a binary variable that is 1 if task t is the last task for sensor s and zero otherwise. Let $y_{s,j}$ be a binary variable that is one if job j is completed by sensor s and zero otherwise. Let y_j be a binary variable that is one if job j is completed and zero otherwise. Let q_s be a binary variable that is one if sensor s is selected and zero otherwise. Each sensor has an associated cost, C_s, and we assume an overall budget of B. Let $\mathbb{T}_{s,j}$ be the set of candidate tasks for sensor s that can perform job j. With these definitions, our MILP for optimal design of a sensor system with both gimbaled and static sensors is presented below.

$$\max \sum_{j \in \mathbb{J}} W_j y_j \tag{1a}$$

s.t.

$$\sum_{s \in \mathbb{S}} C_s q_s \leq B \tag{1b}$$

$$y_j \leq \sum_{s \in \mathbb{S}} y_{s,j} \qquad \forall j \in \mathbb{J} \tag{1c}$$

$$y_{s,j} \leq q_s \qquad \forall j \in \mathbb{J} \ \forall s \in \mathbb{S} \tag{1d}$$

$$y_{s,j} \leq \sum_{t \in \mathbb{T}_{s,j}} x_{s,t} \qquad \forall j \in \mathbb{J} \ \forall s \in \mathbb{S} \tag{1e}$$

$$\sum_{t \in \mathbb{F}_s} f_{s,t} = 1 \qquad \forall s \in \mathbb{S} \tag{1f}$$

$$\sum_{t \in \mathbb{L}_s} l_{s,t} = 1 \qquad \forall s \in \mathbb{S} \tag{1g}$$

$$\sum_{v|(u,v) \in \mathbb{A}_s} z_{s,u,v} = x_{s,u} - l_{s,u} \qquad \forall u \in \mathbb{T}_s \ \forall s \in \mathbb{S} \tag{1h}$$

$$\sum_{u|(u,v) \in \mathbb{A}_s} z_{s,u,v} = x_{s,v} - f_{s,v} \qquad \forall v \in \mathbb{T}_s \ \forall s \in \mathbb{S} \tag{1i}$$

$$0 \leq y_j \leq 1 \qquad \forall j \in \mathbb{J} \tag{1j}$$

$$0 \leq y_{s,j} \leq 1 \qquad \forall j \in \mathbb{J} \ \forall s \in \mathbb{S} \tag{1k}$$

$$x_{s,t} \in \{0,1\} \qquad \forall t \in \mathbb{T}_s \ \forall s \in \mathbb{S} \tag{1l}$$

$$0 \leq z_{s,u,v} \leq 1 \qquad \forall (u,v) \in \mathbb{A}_s \ \forall s \in \mathbb{S} \tag{1m}$$

$$0 \leq f_{s,t} \leq 1 \qquad \forall t \in \mathbb{T}_s \ \forall s \in \mathbb{S} \tag{1n}$$

$$0 \leq l_{s,t} \leq 1 \qquad \forall t \in \mathbb{T}_s \ \forall s \in \mathbb{S} \tag{1o}$$

$$q_s \in \{0,1\} \qquad \forall s \in \mathbb{S} \tag{1p}$$

The objective (equation (1a)) is to maximize the weighted sum of the jobs that get completed. Constraint (1b) ensures that the overall budget is not exceeded. Constraint (1c) ensures that job j is only considered completed if there is at least one sensor that can complete job j. Constraint (1d) ensures that job j is only considered completed by sensor s if sensor s is selected. Similarly, constraint (1e) indicates that sensor s can only complete job j if at least one candidate task for sensor s that can complete job j is selected. Note that the set $\mathbb{T}_{s,j}$ is pre-computed by running simulations to determine which sensors can complete each job. Constraints (1f) and (1g) ensure that each sensor has exactly one first task and one last task, respectively. Constraint (1h) ensures that each selected task is followed by exactly one other task unless it is the last task. Constraint (1i) ensures that each selected task is preceded by exactly one other task unless it is the first task. Constraints (1j) - (1p) restrict the domains of the decision variables.

Note that only $x_{s,t}$ and q_s must be specified as binary. The remaining variables will solve to either zero or one due to properties of the problem. Constraints (1c) - (1e) will force $y_{s,j}$ and y_j to zero if the corresponding right-hand-side of the constraint is zero. Otherwise, the objective will push them to one. Similarly, if all $x_{s,t}$ are set feasibly, then $z_{s,u,v}$, $l_{s,t}$, and $f_{s,t}$ must all be either zero or one. Consider node a. If $x_{s,a}$ is zero, then the right hand side of Constraints (1h) and (1i) are at most zero because $l_{s,a}$ and $f_{s,a}$ are required to be positive. However, the left hand sides of Constraints (1h) and (1i) must be positive because of the bounds on $z_{s,a,v}$. Therefore, both the left and right hand sides must be zero. For this to be true, $l_{s,a}$, $f_{s,a}$, $z_{s,a,v} \ \forall \ v|(a,v) \in \mathbb{A}_s$ (outgoing edges for a), and $z_{s,u,a} \ \forall \ u|(u,a) \in \mathbb{A}_s$ (incoming edges for a) must be zero. Now consider the earliest selected node, b. In other words, $x_{s,b} = 1$ and the task time for node b is earlier than any other selected node. Two situations are possible. Either node b has no incoming edges, or $z_{s,u,b} = 0 \ \forall \ u|(u,b) \in \mathbb{A}_s$ because all of the incoming nodes have $x_{s,u} = 0$. As a result, $f_{s,b} = 1$ by Constraint (1i). According to Constraint (1f), $f_{s,t} = 0$ for any $t \in \mathbb{T}_s$ that is not b. Similarly, if node c is the latest selected node, then $z_{s,c,v}$ is zero for all of the outgoing edges (see argument above) and $l_{s,c} = 1$. Now let node d be the second earliest selected node. We have $z_{s,b,d} = 1$ according to Constraint (1i) because all nodes prior to d except b were not selected. Due to Constraint (1h), $z_{s,b,v} = 0$ except when $v = d$. In other words, exactly one outgoing edge from node b is selected, which is the edge connected to d. A similar argument holds for the next earliest selected node, and so on.

There are many ways to generate candidate tasks for the gimbaled sensors, and the most appropriate approach is likely application dependent. Static sensors can be considered either by using candidate tasks that all have the same pointing direction (which results in a simple graph where every node except the first and last have exactly one incoming and one outgoing edge after the transitive edges are removed) or by pre-computing the set of jobs that may be completed by the sensor and excluding the scheduling constraints for that sensor. In the latter case, Constraints (1d) - (1i) may be replaced by

$$y_{s,j} \leq q_s \qquad \forall j \in \mathbb{C}_s \ \forall s \in \mathbb{S}_s \tag{2}$$

$$y_{s,j} = 0 \qquad \forall j \notin \mathbb{C}_s \tag{3}$$

where $\mathbb{C}_s$ is the set of jobs that can be completed by sensor s and $\mathbb{S}_s$ is the set of static sensors.

If the MILP presented in (1) is solved with a branch and bound (B&B) solver such as Gurobi or HiGHS, the solution provides the sensor system that provably maximizes the objective [10, 11]. The optimal solution accounts for the fact that the gimbaled sensors must be scheduled by embedding the scheduling problem for each sensor within the MILP. Because the MILP is solved with all sensors simultaneously, the schedule for each sensor accounts for the tasks selected for the other sensors. Therefore this MILP can correctly distinguish between the static and gimbaled sensors. The optimal solution may contain all static sensors, all gimbaled sensors, or a mix of static and gimbaled sensors.

Note that the use of the feasibility DAG makes the MILP formulation remarkably generic. This formulation could be directly used for other applications such as constellation design. Most approaches to constellation design maximize geometric coverage [12]. While this is the correct objective for some applications, it is only a proxy objective for others. The MILP presented here could enable the design of constellations targeting specific sets of jobs. In this case, the set $\mathbb{S}$ would include the sensor technology and the candidate orbit instead of a candidate fixed location.

While the MILP presented above is extremely useful, it is computationally challenging to solve. Embedding a scheduling problem within the design problem makes the MILP much larger, and the problem can be intractable when considering both long time horizons and large sets of jobs. In the next section, we describe a decomposition algorithm that can significantly improve computational performance.

4. DECOMPOSITION ALGORITHM

In order to reduce the computational complexity of the MILP presented in the previous section, we partition the overall time horizon (H) into a number of smaller time intervals and use a custom B&B algorithm that solves significantly smaller MILPs representing the portion of the problem corresponding to these time paritions. The B&B algorithm is based on the one proposed in [13] for global solution of stochastic nonlinear optimization problems. We use a customization of this algorithm where we decompose the problem by time instead of scenario.

For the purposes of explanation, suppose we divide H into two time partitions. In order to partition the feasibility DAG, we introduce a number of candidate tasks at the boundary between the two partitions. We denote these candidate tasks as boundary tasks. The boundary tasks do not necessarily complete any jobs. Instead, they represent the pointing direction of the sensor at the boundary time. We use these candidate tasks to partition the graph into two parts. The last task for the first graph must be the first task of the second graph, both of which must be one of the boundary tasks. If enough boundary tasks are included (i.e., the space of pointing directions is discretized finely enough), the optimal solution will not change. Using these concepts, we rewrite the MILP from Section 3 as follows.

$$\max \sum_{p\in\mathbb{P}} \sum_{j\in\mathbb{J}_p} W_j y_{j,p} \tag{4a}$$

s.t.

$$\sum_{s\in\mathbb{S}} C_s q_{s,p} \le B \qquad \forall p \in \mathbb{P} \tag{4b}$$

$$q_s = q_{s,p} \qquad \forall p \in \mathbb{P}\ \forall s \in \mathbb{S} \tag{4c}$$

$$y_{j,p} \le \sum_{s\in\mathbb{S}} y_{s,j,p} \qquad \forall j \in \mathbb{J}_p\ \forall p \in \mathbb{P} \tag{4d}$$

$$\sum_{p\in\mathbb{P}_j} y_{j,p} \le 1 \qquad \forall j \in \mathbb{J} \tag{4e}$$

$$y_{s,j,p} \le q_{s,p} \qquad \forall j \in \mathbb{J}_p\ \forall p \in \mathbb{P}\ \forall s \in \mathbb{S} \tag{4f}$$

$$y_{s,j,p} \le \sum_{t\in\mathbb{T}_{s,j,p}} x_{s,p,t} \qquad \forall j \in \mathbb{J}_p\ \forall p \in \mathbb{P}\ \forall s \in \mathbb{S} \tag{4g}$$

$$\sum_{t\in\mathbb{F}_{s,p}} f_{s,p,t} = 1 \qquad \forall s \in \mathbb{S}\ \forall p \in \mathbb{P} \tag{4h}$$

$$\sum_{t\in\mathbb{L}_{s,p}} l_{s,p,t} = 1 \qquad \forall s \in \mathbb{S}\ \forall p \in \mathbb{P} \tag{4i}$$

$$f_{s,p,t} = l_{s,p-1,t} \qquad \forall t \in \mathbb{F}_{s,p} \ \forall p \in \{i \in \mathbb{P} | i > 1\} \ \forall s \in \mathbb{S} \tag{4j}$$

$$\sum_{v|(u,v)\in\mathbb{A}_{s,p}} z_{s,p,u,v} = x_{s,p,u} - l_{s,p,u} \qquad \forall u \in \mathbb{T}_s \ \forall p \in \mathbb{P} \ \forall s \in \mathbb{S} \tag{4k}$$

$$\sum_{u|(u,v)\in\mathbb{A}_{s,p}} z_{s,p,u,v} = x_{s,p,v} - f_{s,p,v} \qquad \forall v \in \mathbb{T}_s \ \forall p \in \mathbb{P} \ \forall s \in \mathbb{S} \tag{4l}$$

$$0 \le y_{j,p} \le 1 \qquad \forall j \in \mathbb{J}_p \ \forall p \in \mathbb{P} \tag{4m}$$

$$0 \le y_{s,j,p} \le 1 \qquad \forall j \in \mathbb{J}_p \ \forall p \in \mathbb{P} \ \forall s \in \mathbb{S} \tag{4n}$$

$$x_{s,p,t} \in \{0,1\} \qquad \forall t \in \mathbb{T}_{s,p} \ \forall p \in \mathbb{P} \ \forall s \in \mathbb{S} \tag{4o}$$

$$0 \le z_{s,p,u,v} \le 1 \qquad \forall (u,v) \in \mathbb{A}_{s,p} \ \forall p \in \mathbb{P} \ \forall s \in \mathbb{S} \tag{4p}$$

$$0 \le f_{s,p,t} \le 1 \qquad \forall t \in \mathbb{T}_{s,p} \ \forall p \in \mathbb{P} \ \forall s \in \mathbb{S} \tag{4q}$$

$$0 \le l_{s,p,t} \le 1 \qquad \forall t \in \mathbb{T}_{s,p} \ \forall p \in \mathbb{P} \ \forall s \in \mathbb{S} \tag{4r}$$

$$q_{s,p} \in \{0,1\} \qquad \forall p \in \mathbb{P} \ \forall s \in \mathbb{S} \tag{4s}$$

$$q_s \in \{0,1\} \qquad \forall s \in \mathbb{S} \tag{4t}$$

Here, $\mathbb{P}$ is the set of time partitions used for decomposition, $\mathbb{P}_j$ is the set of partitions that have at least partial overlap with the time interval for job j, and $\mathbb{J}_p$ is the set of jobs with time intervals that at least partially overlap with partition p. Figures 2 and 3 illustrate the last two sets. In Figure 2, the time interval in which the job must be completed lies completely within a single partition. In this case, $|\mathbb{P}_j| = 1$, $\mathbb{P}_j = \{2\}$, $j \notin \mathbb{J}_1$, $j \in \mathbb{J}_2$, and $j \notin \mathbb{J}_3$. In Figure 3, the time interval in which the job must be completed crosses a partition boundary. In this case, $|\mathbb{P}_j| = 2$, $\mathbb{P}_j = \{2,3\}$, $j \notin \mathbb{J}_1$, $j \in \mathbb{J}_2$, and $j \in \mathbb{J}_3$.

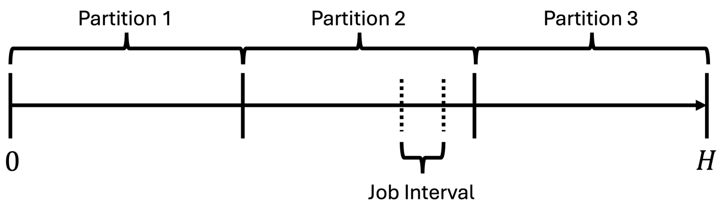

Fig. 2: Example job interval that lies completely within one time partition ($|\mathbb{P}_j| = 1$).

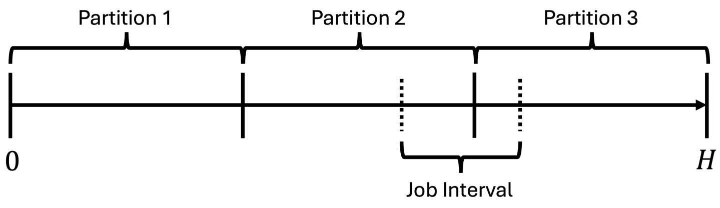

Fig. 3: Example job interval that crosses a partition boundary and, therefore, overlaps with two partitions ($|\mathbb{P}_j| = 2$).

Formulation (4) is mostly the same as (1) except that it has additional indices for many of the variables and sets to indicate which time partition each is associated with. However, we have included three additional constraints: (4c), (4e), and (4j). Constraint (4c) ensures that the same sensors are selected across all partitions. Note that $y_{j,p}$ is a binary variable indicating whether job j is completed in partition p. Constraint (4e) ensures that a job can only be completed once even if the interval for that job overlaps with multiple time partitions. Constraint (4j) forces the last task selected for one partition to be the first task selected for the following partition.

If we drop constraints (4c), (4e), and (4j), problem (4) completely decomposes by partition and can be solved in parallel. Furthermore, because we have only modified the problem by dropping constraints, we have created a relaxation of the problem. Every feasible solution for the original problem is feasible for the relaxation. As a result, the solution to the relaxation gives an upper bound on the optimal objective value. On the other hand, any feasible solution to problem

(4) gives a lower bound to the optimal objective value. This idea provides the basis of our custom B&B algorithm which is outlined in Algorithm 1.

The algorithm starts by solving the root node of the branch and bound tree in parallel (line 1). The root node is constructed by dropping Constraints (4c), (4e), and (4j). When these constrains are removed, the problem decomposes by partition. As a result, we may solve the node by solving the MILP for each partition independently (and in parallel). The MILP for a partition is the same as Problem (4), except $|\mathbb{P}| = 1$. To compute the objective for the node, f_n, we sum the objectives obtained for each partition. As the algorithm proceeds, we ensure the problem remains separable by partition.

Like any B&B algorithm, our algorithm proceeds by computing lower (L) and upper (U) bounds on the optimal objective. We then attempt to increase the lower bound and decrease the upper bound until the optimal solution is found. Because our problem is a maximization problem, the lower bound at any point in the algorithm is the objective value of the best feasible solution found so far. The upper bound is initialized to the objective value of the root node, f_{n_0}. The upper bound is decreased by branching on a variable or a set of variables (lines 18 - 32). The process of branching creates new nodes in the B&B tree. Branching is done in a way that guarantees that the objective of each node is less than or equal to the objective of its parent node. Furthermore, the optimal solution ($y^*_{j,p}$, $y^*_{s,j,p}$, $x^*_{s,p,t}$, $z^*_{s,p,u,v}$, $f^*_{s,p,t}$, $l^*_{s,p,t}$, and $q^*_{s,p}$) at a node should be infeasible for all of its descendant nodes.

The set of unexplored nodes in the B&B tree, $\mathbb{N}$, is initialized with the root node on line 2. The lower bound is initialized to negative infinity on line 3 because no feasible solution has been found yet. The upper bound is initialized to the objective of the root node on line 4. The main loop of the algorithm starts on line 5, and the algorithm continues until the lower and upper bounds are sufficiently close. On line 6, we select the next node to be explored by popping the node with the largest objective from the set of unexplored nodes. The node with the largest objective is selected with the hope that this will reduce the upper bound.

On lines 7 - 10, we check if the solution found when solving the selected node satisfies Constraints (4c), (4e), and (4j). All other constraints are guaranteed to be satisfied because they are included in each subproblem solved on lines 1, 33, and 34. Therefore, if the solution for a node satisfies Constraints (4c), (4e), and (4j), then all constraints in Problem (4) have been satisfied, and a feasible solution has been found. Furthermore, this solution is the optimal solution and the algorithm can terminate. This is guaranteed because, at this point, $f_n = U$ and U is always greater than or equal to L. This will be explained more below. If this is the case, we update the lower bound and optimal solution on lines 8 and 9, respectively, before exiting the while loop on line 10.

If the solution for node n is not feasible, then we use a heuristic to find a feasible solution at node n on line 12. The heuristic ensures that the feasible solution found abides by any restrictions imposed on the node through the branching process. Our heuristic begins by computing the average value of $q_{s,p}$ found at the solution of the node for each sensor across all partitions:

$$\overline{q_s} = \frac{1}{|\mathbb{P}|} \sum_{p \in \mathbb{P}} q_{s,p} \qquad \forall s \in \mathbb{S} \tag{5}$$

We select a subset of the sensors, $\hat{\mathbb{S}} \subseteq \mathbb{S}$, such that $\sum_{s \in \hat{\mathbb{S}}} \overline{q_s}$ is maximized while constraining $\sum_{s \in \hat{\mathbb{S}}} C_s \leq B$. For each s in $\hat{\mathbb{S}}$, we fix $q_{s,p} = 1$. For each s not in $\hat{\mathbb{S}}$, we fix $q_{s,p} = 0$. Next, we relax $x_{s,p,t}$ and $q_{s,p}$ to be continuous variables between zero and one instead of binary variables so that the next sequence of problems solved in the heuristic are linear programs (LPs) instead of MILPs. Then, we solve the problem with these changes applied, check for jobs where Constraint (4e) is violated, fix $y_{j,p}$ to zero for all partitions except one, and repeat until Constraint (4e) is satisfied for all jobs. We fix $y_{j,p}$ for ten jobs each iteration. At this point, we continue to iteratively solve LP relaxations while fixing $f_{s,p,t}$ and $l_{s,p,t}$ so that Constraint (4j) is satisfied. Finally, we make $x_{s,p,t}$ binary again and solve the resulting MILP. This provides a feasible solution and completes the heuristic. After the heuristic is complete, we check if the feasible solution found is better than the best feasible solution found so far. If so, we update L and q^*_s on lines 14 and 15, respectively.

After the heuristic is finished, we begin the branching process on line 18 by creating two new nodes that inherit all restrictions (generated on lines 22, 26, and/or 30-31) applied to any of their ancestors (i.e., the new nodes are "copied" from their parent node). We prioritize Constraint (4c) and then Constraint (4e) for branching. If we find any sensor where Constraint (4c) is violated (line 19), we select the sensor whose average $q_{s,p}$ value is closest to 0.5 for branching (lines 20 - 21). For the selected sensor, we fix $q_{\tilde{s},p}$ to zero for all partitions in node n_1 and to one for all partitions in

Algorithm 1 Parallel branch and bound algorithm for solving Problem (4). Given a convergence tolerance, ε, and the root node, n_0, which is the relaxation of problem (4) generated by dropping Constraints (4c), (4e), and (4j). Produces the optimal sensor system, q_s^* and the optimal objective value, L.

```
 1: Solve $n_0$ in parallel and set $f_{n_0}$ to the optimal objective value
 2: $\mathbb{N} \leftarrow \{n_0\}$
 3: $L \leftarrow -\infty$
 4: $U \leftarrow f_{n_0}$
 5: while $\frac{(U-L)}{L} > \varepsilon$ do
 6:     Pop node $n$ from $\mathbb{N}$ with the largest objective, $f_n$
 7:     if constraints (4c), (4e), and (4j) are feasible at the optimal solution for node $n$ then
 8:         $L \leftarrow f_n$
 9:         Let $q_s^*$ be the values of $q_s$ at the optimal solution for node $n$
10:         break
11:     else
12:         Use a heuristic to find a feasible solution for node $n$; let $g_n$ be the objective value, and $\hat{q}_s$ be the values of $q_s$
            selected by the heuristic
13:         if $g_n > L$ then
14:             $L \leftarrow g_n$
15:             $q_s^* \leftarrow \hat{q}_s$
16:         end if
17:     end if
18:     Create two new nodes, $n_1$ and $n_2$, both copied from node $n$
19:     if Constraint (4c) is violated then
20:         Let $\overline{q_s} = \frac{1}{|\mathbb{P}|}\sum_{p\in\mathbb{P}} q_{s,p}$
21:         Select $\tilde{s} = \arg\min_{s\in\mathbb{S}} |\overline{q_s} - 0.5|$
22:         Fix $q_{\tilde{s},p} = 0$ in $n_1$ and $q_{\tilde{s},p} = 1$ in $n_2$.
23:     else if Constraint (4e) is violated then
24:         Select $\tilde{j} = \arg\max_{j\in\mathbb{J}} \sum_{p\in\mathbb{P}} y_{j,p}$
25:         Split $\mathbb{P}_{\tilde{j}}$ into two sets, $\mathbb{P}_1$ and $\mathbb{P}_2$ such that $\mathbb{P}_1 \cup \mathbb{P}_2 = \mathbb{P}_{\tilde{j}}$, $\mathbb{P}_1 \cap \mathbb{P}_2 = \emptyset$, and $\sum_{p\in\mathbb{P}_1} y_{j,p} \approx \sum_{p\in\mathbb{P}_2} y_{j,p}$
26:         Fix $y_{j,p} = 0 \ \forall p \in \mathbb{P}_1$ in node $n_1$ and $y_{j,p} = 0 \ \forall p \in \mathbb{P}_2$ in node $n_2$
27:     else
28:         Select a sensor-partition pair ($\tilde{s}$, $\tilde{p}$) where Constraint (4j) is violated.
29:         Create a plane ($c = ax + by + cz + d = 0$) that separates the pointing directions associated with the last task
            selected for partition $\tilde{p} - 1$ and the first task selected for partition $\tilde{p}$.
30:         Fix $l_{\tilde{s},\tilde{p}-1,t} = 0$ and $f_{\tilde{s},\tilde{p},t} = 0$ for every task with a pointing direction on one side of the plane ($c > 0$) in node
            $n_1$.
31:         Fix $l_{\tilde{s},\tilde{p}-1,t} = 0$ and $f_{\tilde{s},\tilde{p},t} = 0$ for every task with a pointing direction on the other side of the plane ($c < 0$) in
            node $n_2$.
32:     end if
33:     Solve $n_1$ in parallel and set $f_{n_1}$ to the optimal objective value
34:     Solve $n_2$ in parallel and set $f_{n_2}$ to the optimal objective value
35:     $\mathbb{N} \leftarrow \mathbb{N} \cup \{n_1, n_2\}$
36:     $U \leftarrow \max_{n\in\mathbb{N}} f_n$
37: end while
38: return $q_s^*$, $L$
```

node n_2. If Constraint (4c) is satisfied, then we check for jobs where Constraint (4e) is violated. If there are multiple jobs where Constraint (4e) is violated, we select the job, $\tilde{j}$, where Constraint (4e) is violated by the largest amount (line 24). We then split the set of partitions that overlap with the time interval for job $\tilde{j}$ into two sets as described on line 25. For nodes n_1, we fix $y_{j,p}$ to zero for all partitions in the first set. For node n_2, we fix $y_{j,p}$ to zero for all partitions in the second set. If Constraints (4c) and (4e) are both satisfied, then Constraint (4j) must be violated. We select a sensor-partition pair on line 28 where Constraint (4j) is violated. If the constraint is violated for multiple sensor-partition pairs, we select the pair that has the largest discrepancy in the pointing directions selected by the last task of the previous partition and the first task of the current partition. Let $\vec{v}_1$ and $\vec{v}_2$ be the unit vectors describing the pointing directions associated with the last task selected in partition $\tilde{p}-1$ and the first task selected in partition $\tilde{p}$, respectively. We create a plane to split the space of pointing directions in two (not necessarily equal) parts. The origins of $\vec{v}_1$ and $\vec{v}_2$ both lie on the plane, and the distance from the end of $\vec{v}_1$ to the plane equals the distance from the end of $\vec{v}_2$ to the plane. For node n_1, we set $l_{\tilde{s},\tilde{p}-1,t} = 0$ and $f_{\tilde{s},\tilde{p},t} = 0$ for every task with a pointing direction on the same side of the plane as $\vec{v}_1$. For node n_2, we set $l_{\tilde{s},\tilde{p}-1,t} = 0$ and $f_{\tilde{s},\tilde{p},t} = 0$ for every task with a pointing direction on the same side of the plane as $\vec{v}_2$. This completes the branching process.

After creating the new nodes n_1 and n_2, the MILPs defined by each node are solved on lines 33 - 34. The new nodes are then added to the set of unexplored nodes on line 35. Finally, the upper bound is updated on line 36 based on the maximum objective value of all unexplored nodes before processing another node on line 6.

By using a decomposition algorithm that solves a sequence of smaller MILPs, we can exploit the problem structure while still utilizing commercial or open-source MILP solvers (e.g., Gurobi or HiGHS) for the subproblems. These solvers are extremely effective at handling binary variables and have advanced B&B algorithms that integrate presolve and cutting planes [14–16]. Our decomposition algorithm does not need to worry about binary variables and, instead, only needs to ensure consistency between the different partitions. In the following section, we present computational results for a number of test problems that demonstrate the benefit of Algorithm 1.

5. COMPUTATIONAL RESULTS

Here, we present computational results demonstrating the benefit of the decomposition algorithm described in Section 4. We constructed a number of test problems with varying numbers of jobs, job interval sizes, partition sizes, and budgets. Our test problems considered two sensor technologies. The first is a gimbaled sensor with cost 1. The second is a static sensor that is more capable but has cost 2. We consider 20 candidate locations for every problem. Note that it is possible for the optimal solution to contain a gimbaled sensor and a static sensor in the same location. For all problems, we consider a time horizon of 500 time units. We consider budgets of 3, 5, and 7. For the largest budget, there are 95,534 possible sensor system designs (i.e., q_s), even without considering the scheduling problem. We consider partition intervals of 25 time units and 50 time units and job intervals with 25 time units and 50 time units. Therefore, we have test problems where the job interval is less than, equal to, and greater than the partition interval. We consider sets of jobs ranging from 2,000 jobs to 10,000 jobs. The jobs are split evenly across the overall time horizon. For example, the test problem with 2000 jobs and job intervals of 50 time units has 200 jobs that must be completed within the first 50 time units, 200 jobs that must be completed between time 50 and 100, and so on. As a result, when the length of the job interval equals the length of the partition interval, the two intervals align, and Constraint (4e) will never be violated because $|\mathbb{P}_j| = 1$ for all jobs.

We implemented our algorithm in Python and used Pyomo to model the MILPs [17]. We used Gurobi to solve all MILPs. All computational experiments were run on a linux server with 32 3.6 GHz cores. We set a 1-hour time limit for each test problem. The results are presented in Table 1. Here, "FS" represents the full-space approach where Problem (4) is solved directly with Gurobi without any decomposition. "D" represents the decomposition algorithm described in Section 4. The "Bound" column shows the upper bound (U) obtained by the 1-hour limit. The "Incumbent" column shows the objective of the best feasible solution (L) found after 1 hour. The optimality gap is computed as $\frac{100(U-L)}{L}$. Each row shows the results for a different test problem. For the decomposition algorithm, we limited Gurobi to three threads for each subproblem. For the full-space approach, we did not limit the number of threads Gurobi could use.

The results clearly show the benefit of the decomposition algorithm. The decomposition algorithm outperforms the full-space approach for all problems with more than 5,000 jobs. For one problem, the optimality gap was reduced from 429.6% to 1.6%. For many problems, Gurobi did not even find a feasible solution within the time limit for the

Table 1: Computational results comparing the decomposition algorithm described in Section 4 (denoted "D" in the table) and the full-space approach (dentoed "FS" in the table) where Problem (4) is solved directly with Gurobi without any decomposition. The "Bound" column shows the upper bound obtained after 1 hour. The "Incumbent" column shows the objective of the best feasible solution found after 1 hour. The "Gap (%)" column shows the optimality gap after 1 hour, computed as $\frac{100(U-L)}{L}$.

# Jobs	Job Interval	Partition Interval	Budget	Bound		Incumbent		Gap (%)	
				FS	D	FS	D	FS	D
2000	50	25	3	1013	1130	985	911	2.8%	24.0%
2000	50	25	5	1652	1812	1451	1345	13.9%	34.7%
2000	50	25	7	1818	2376	1736	1559	4.7%	52.4%
2000	50	50	3	1019	1005	992	980	2.7%	2.6%
2000	50	50	5	1564	1514	1453	1432	7.6%	5.7%
2000	50	50	7	1822	1812	1733	1704	5.1%	6.3%
3000	50	25	3	2037	1623	1320	1322	54.3%	22.8%
3000	50	25	5	2724	2697	2094	2057	30.1%	31.1%
3000	50	25	7	2916	3597	2600	2546	12.2%	41.3%
3000	50	50	3	1985	1361	1353	1349	46.7%	0.9%
3000	50	50	5	2682	2154	2079	1653	29.0%	30.3%
3000	50	50	7	2920	2770	2374	2371	23.0%	16.8%
4000	25	25	3	1163	1195	1129	1088	3.0%	9.8%
4000	25	25	5	2313	1898	1714	1710	34.9%	11.0%
4000	25	25	7	2894	2495	2156	2121	34.2%	17.6%
4000	25	50	3	1173	1155	1138	1111	3.1%	4.0%
4000	25	50	5	2073	1850	1769	1735	17.2%	6.6%
4000	25	50	7	2890	2429	2197	2150	31.5%	13.0%
5000	50	25	3	4019	2623	T/O	2205	T/O	19.0%
5000	50	25	5	4771	4559	T/O	3484	T/O	30.9%
5000	50	25	7	4933	6589	T/O	3676	T/O	79.2%
5000	50	50	3	3846	2298	1441	2250	166.9%	2.1%
5000	50	50	5	4745	4448	T/O	3516	T/O	26.5%
5000	50	50	7	4931	4883	T/O	3945	T/O	23.8%
6000	25	25	3	2449	1694	1420	1640	72.5%	3.3%
6000	25	25	5	3832	2752	2201	2699	74.1%	2.0%
6000	25	25	7	4639	3735	2664	3120	74.1%	19.7%
6000	25	50	3	2432	1679	1551	1656	56.8%	1.4%
6000	25	50	5	3736	2775	2166	2707	72.5%	2.5%
6000	25	50	7	4549	3995	2483	3120	83.2%	28.0%
10000	25	25	3	5116	2673	966	2631	429.6%	1.6%
10000	25	25	5	7158	4945	T/O	4432	T/O	11.6%
10000	25	25	7	8300	7502	T/O	5457	T/O	37.5%
10000	25	50	3	5130	2749	1562	2632	228.4%	4.4%
10000	25	50	5	7187	6213	T/O	3301	T/O	88.2%
10000	25	50	7	8323	7993	T/O	4484	T/O	78.3%

full-space problem. For smaller problems, decomposition is less beneficial, but significant improvements can still be seen for some test problems. For example, the decomposition algorithm reduced the optimality gap from 34.9% to 11.0% for the test problem with 4,000 jobs, job intervals equal to 25 time units, partition intervals equal to 25 time units, and a budget of 5.

On the other hand, the results show that the decomposition algorithm performs better when the job interval is shorter than the partition interval. To illustrate this more clearly, we solved a sequence of problems where, within a single problem, some jobs had job intervals of 25 time units and some jobs had job intervals of 50 time units. We fixed the total number of jobs to 10,000 and varied the number of jobs with job intervals of 25. The results are shown in Figure 4. The x-axis shows the number of jobs with time intervals of 25 time units, n_{25}, and the y-axis shows the optimality gap. The decomposition algorithm works better as n_{25} increases, as shown by the decreasing optimality gap. The reason for this is that, when the job intervals are shorter, $|\mathbb{P}_j|$ is smaller, and Constraint (4e) is less likely to be violated at a given node in the B&B tree.

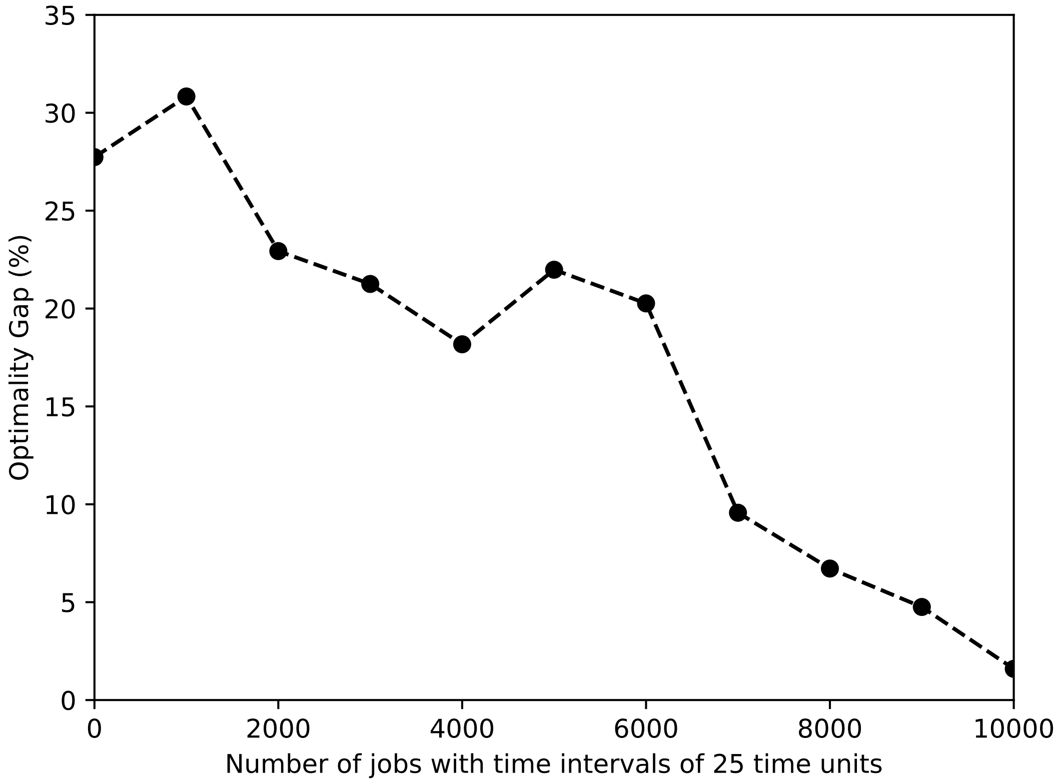

Fig. 4: Effect of job interval on the decomposition algorithm. The test problem has a budget of 3 and a partition interval of 25. The number of jobs with time intervals of 25, n_{25}, is shown on the x-axis. The total number of jobs is 10,000. The resulting optimality gap for each problem is shown on the y-axis. For a fixed partition interval, the optimality gap improves as the average job interval gets shorter because $|\mathbb{P}_j|$ decreases.

6. CONCLUSION

We presented a mixed-integer linear programming (MILP) formulation to optimally design a sensor system for space situational awareness, including selection of both sensor technology and sensor location. The MILP formulation properly distinguishes between static and gimbaled sensors by embedding a scheduling problem for each gimbaled sensor in the MILP. Additionally, the MILP formulation is remarkably generic and can be applied to related problems such as constellation design. Because the MILP can be computationally challenging for large numbers of jobs and long time horizons, we presented a decomposition algorithm that partitions the overall time horizon into a number of smaller intervals. As a result, we can solve a number of smaller MILPs within a parallel branch-and-bound (B&B) algorithm. The B&B algorithm ensures consistency between the different time partitions (e.g., each partition needs to select the same sensors). Our computational results showed that the decomposition algorithm can significantly outperform the full-space approach, where the original MILP is solved directly without any decomposition. Additionally, we found that the decomposition algorithm performs better when the majority of the job intervals lie entirely within a single partition. Future work could consider extending the MILP formulation to a stochastic programming problem, which would enable consideration of multiple sets of jobs and provide a design that is more reliable across scenarios not considered directly in the optimization problem.

7. ACKNOWLEDGEMENTS

We would like to thank Cindy Phillips for helping to prove that $z_{s,u,v}$, $f_{s,t}$, and $l_{s,t}$ do not need to be binary variables.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

8. REFERENCES

[1] SW Legg, AJ Benavides-Serrano, John D Siirola, Jean-Paul Watson, SG Davis, Are Bratteteig, and Carl D Laird. A stochastic programming approach for gas detector placement using cfd-based dispersion simulations. *Computers & Chemical Engineering*, 47:194–201, 2012.

[2] Jonathan Berry, William E Hart, Cynthia A Phillips, James G Uber, and Jean-Paul Watson. Sensor placement in municipal water networks with temporal integer programming models. *Journal of water resources planning and management*, 132(4):218–224, 2006.

[3] Ashley D Biria and Belinda G Marchand. Constellation design for space-based space situational awareness applications: an analytical approach. *Journal of Spacecraft and Rockets*, 51(2):545–562, 2014.

[4] Yuri Ulybyshev. Satellite constellation design for complex coverage. *Journal of Spacecraft and Rockets*, 45(4): 843–849, 2008.

[5] Ali Ahmadzadeh, James Keller, George Pappas, Ali Jadbabaie, and Vijay Kumar. An optimization-based approach to time-critical cooperative surveillance and coverage with uavs. In *Experimental robotics: The 10th international symposium on experimental robotics*, pages 491–500. Springer, 2008.

[6] Katherine A Klise, Bethany L Nicholson, Carl D Laird, Arvind P Ravikumar, and Adam R Brandt. Sensor placement optimization software applied to site-scale methane-emissions monitoring. *Journal of Environmental Engineering*, 146(7):04020054, 2020.

[7] Jianfeng Liu and Carl D Laird. A global stochastic programming approach for the optimal placement of gas detectors with nonuniform unavailabilities. *Journal of Loss Prevention in the Process Industries*, 51:29–35, 2018.

[8] AJ Benavides-Serrano, MS Mannan, and CD Laird. Optimal placement of gas detectors: Ap-median formulation considering dynamic nonuniform unavailabilities. *AIChE Journal*, 62(8):2728–2739, 2016.

[9] Xinwei Wang, Guohua Wu, Lining Xing, and Witold Pedrycz. Agile earth observation satellite scheduling over 20 years: Formulations, methods, and future directions. *IEEE Systems Journal*, 15(3):3881–3892, 2020.

[10] Qi Huangfu and JA Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018.

[11] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL `https://www.gurobi.com`.

[12] Martin W Lo. Satellite-constellation design. *Computing in science & engineering*, 1(01):58–67, 1999.

[13] Yankai Cao and Victor M Zavala. A scalable global optimization algorithm for stochastic nonlinear programs. *Journal of Global Optimization*, 75(2):393–416, 2019.

[14] Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2):473–506, 2020.

[15] Ralph Edward Gomory. An algorithm for the mixed integer problem. *Report No. P-1885, The Rand Corporation, Santa Monica, CA.*, 1960.

[16] Hugues Marchand and Laurence A Wolsey. Aggregation and mixed integer rounding to solve mips. *Operations research*, 49(3):363–371, 2001.

[17] Michael L Bynum, Gabriel A Hackebeil, William E Hart, Carl D Laird, Bethany L Nicholson, John D Siirola, Jean-Paul Watson, David L Woodruff, et al. *Pyomo-optimization modeling in python*, volume 67. Springer, 2021.