Nonoverlapping Grid-aligned Rectangle Placement for High Value Areas

Stephen Rowe* Christopher G. Valicka† Scott A. Mitchell‡ Simon X. Zou‡

Abstract

We consider heuristic and optimal solutions to a discrete geometric bin packing problem that arises in a resource allocation problem. An imaging sensor is assigned to collect data over a large area, but some subregions are more valuable than others. To capture these high-value regions with higher fidelity, we can assign some number of non-overlapping rectangular subsets, called "subfootprints." The sensor image is partitioned into squares called "chips", and each chip is further partitioned into pixels. Pixels may have different values. Subfootprints are restricted to rectangular collections of chips, but we are free to choose different rectangle heights, widths, and areas. We seek the optimal arrangement over the family of possible rectangle shapes and sizes.

We provide a mixed-integer linear program optimization formulation, as well as a greedy heuristic, to solve this problem. For the meta-problem, we have some freedom to align the chip boundaries to different pixels. However, it is too expensive to solve the optimization formulation for each alignment. However, we show that the greedy heuristic can inform which pixel alignments are worth solving the optimization over. We use a variant of k-means clustering to group greedy solutions by their transport shape-similarity. For each cluster, we run the optimization problem over the greedy layout with the highest value. In practice this efficiently explores the geometric configuration space, and produces solutions close to the global optimum. We show a contrived example using surveillance of the Mississippi River. Our software is available as open-source in the Github repository "GeoPlace."

1 Introduction

We study a geometric bin packing problem that arises in bandwidth-constrained sensing, e.g. satellite-based sensors. A camera, or other sensor, observes a region of interest. The image, called a *footprint*, is transmitted to some earth-based analyst for downstream processing. The capacity to capture new images exceeds the transmission capacity, so we must make some choices about what data to transmit. Further, some pixels of the image are more interesting than others to the analyst, so it makes sense to devote extra bandwidth to those. Sensor systems are often designed with the capability to devote extra bandwidth (or resolution) to selected subareas, called *sub-footprints*.

However, we are not free to select an arbitrary set of pixels, but are limited by the hardware and software design of the system. A common design for satellite-borne sensors is to partition the pixels of the footprint into a regular rectangular grid of *chips*, each comprised of the same number of pixels. Further, they have the capability to select sub-footprints that are rectangular sets of chips. Each sub-footprint is allocated the same bandwidth, rather than each chip, so typically sub-footprints of about the same area are desired, but this is not a constraint. Sub-footprints are constrained not to overlap, but we may select the width and height of each one independently. That is, the sub-footprints are not required to be translations or rotations of a single shape. The system supports a maximum number of subfootprints. Our objective is to optimally choose rectangular subfootprints that cover the pixels of particular interest, while making efficient use of of the available bandwidth.

Existing techniques for solving this problem usually involve some manual placement of the sub-footprints by an experienced human, someone who would rather be considering higher-level questions. Also, real-time situations arise in which human intervention is impractical. In practice, data collection scenarios are planned hours or more in advance and planners may choose near-optimal sub-footprint placements. At collection time, however, target and spacecraft conditions usually create a difference between planned and actual sensor pointing, external information changes pixel priorities, or both. Accordingly, optimal sub-footprint placements are often different from planned placements and the time required for operators to update solutions is longer than the time available to upload solutions to the spacecraft for execution. Quantifying satellite sensor time can be ambiguous as satellite missions differ, as do the costs of different satellite platforms. In scientific applications, timely information regarding weather or sea-state can have implications regarding the safety of critical infrastructure and human lives. In commercial applications, missed or poorly chosen footprint placements can result in reduced revenue.

Algorithmic solutions to related problems in the satellite sensor planning and tasking literature are limited. Song et al. [6] present algorithms to solve the Satel-

*Systems Mission Engineering, Sandia National Laboratories.
†Information Systems Analysis Center,
‡Center for Computing Research, samitch@sandia.gov,

lite Frame Selection (SFS) problem: for each satellite imaging opportunity, a rectangular satellite footprint is selected to maximally balance coverage of a set of client imaging requests with the resolution requested. Often, sensor placement problems assume circular or elliptical sensor footprints that are allowed to overlap when determining coverage [7], and many placement problems focus on collection scheduling problems [5, 3, 9] with less emphasis on the geometric aspects of the problem.

However, the computational geometry community has considered similar problems arising from other contexts. Packing rectangles into integer grids for efficient device layouts on semiconductor wafers [1] appears related to selecting our sub-footprints. Shifting an integer grid so that many cells contain valuable points [2] appears related to shifting our footprint by a few pixels for improved sub-footprint solutions. By demonstrating a geometric solution to the "sub-footprint" problem, we hope to inspire the computational geometry community to generate novel solutions for satellite planning.

2 "Sub-footprint" Problem Statement

The image footprint is defined as a Cartesian grid of $M \times N$ pixels, which is partitioned into a coarser grid of $M_C \times N_C$ chips, where each chip is comprised of $\Delta x \times \Delta y$ pixels. We must select sub-footprints that are rectangular arrays of chips. The chips are in fixed position relative to the footprint, but it is possible to shift the entire footprint grid by several pixels for better pixel-to-chip alignment. The number of sub-footprints is K, where $K \leq K_{\max}$. The available bandwidth is B. The bandwidth allocated to each sub-footprint is equal, regardless of the number of chips A^k it contains. Consequently, a chip in sub-footprint k receives B/KA^k bandwidth.

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. The value of a sub-footprint is the sum of the value of its chips, and the value of the solution is the sum of sub-footprint values. Our goal is to choose the highest value sub-footprints, constrained by the bandwidth. (We also consider the problem of minimizing necessary bandwidth subject to the constraint that all valuable pixels are captured.) One could choose a single large sub-footprint to cover many chips, but the disadvantages are that the bandwidth per chip is low, and because the sub-footprint is rectangular it is likely to cover many low-priority chips. On the other hand, one could choose many small sub-footprints to cover the chips of interest, but there is an upper bound on the number of sub-footprints, and they must not overlap, so it may not be feasible to capture all the high value chips or use the bandwidth efficiently.

Solutions based on optimization formulations are often preferred in this context because they may be seamlessly coupled to a larger optimization-based scheduling framework. However, objectives are often not crisply defined, and finding an exactly optimal solution is often overkill. Solvers tend to be orders of magnitude slower than geometric algorithms. Our solution includes an optimization formulation (Section 3), but we use geometric algorithms to propose a tractable set of scenarios for the optimizer (Section 4).

3 Mixed-Integer Problem

We seek a mixed-integer linear constrained optimization problem (MIP) description that can be efficiently solved by current solvers. The model should try to simultaneously cover high priority chips, give high priority chips the most bandwidth possible, and guarantee that the available bandwidth is not exceeded.

As a secondary objective, we prefer to give higher priority chips more bandwidth; consequently, if two solutions both include a chip with a high priority, we prefer the solution where it is in a smaller sub-footprint. We model this with an objective-function penalty-term proportional to sub-footprint area.

3.1 Footprint Position Constraints, Chip Definitions

We let S^k denote the k-th sub-footprint, and $C^{i,j}$ refer to the chip in the ith row and jth column of the image. We begin by enforcing that the coordinates of a sub-footprint's lower left corner S_0 are not more than the coordinates of its upper right corner S_1. By "coordinate," we mean its chip index: the lower left chip $C^{0,0}$ of the image lies at $(0, 0)$, and the upper right at $(M_C - 1, N_C - 1)$. A sub-footprint covering a single chip has $S_0 = S_1$. Let S^k_{0x} denote the variable for the x coordinate, etc. We constrain

$$S_{0x} \leq S_{1x}, \quad S_{0y} \leq S_{1y}. \tag{1}$$

3.2 Non-Overlapping Rectangles

We label sub-footprints based on the lexicographic order of their lower corner. This reduces the optimization solve time significantly by eliminating symmetric solutions, since sub-footprints are equivalent under relabelling. By "lexicographic order" we mean $(x_0, y_0) \leq (x_1, y_1)$ if and only if $y_0 < y_1$ or $y_0 = y_1$ and $x_0 \leq x_1$. We enforce this by the following constraint $\forall k$:

$$S^k_{0x} + M_C \cdot S^k_{0y} \leq S^{k+1}_{0x} + M_C \cdot S^{k+1}_{0y} \tag{2}$$

We start by ensuring rectangles do not overlap by disjunctive constraints: S^k must be completely to the right of S^l, or S^k must be above S^l, or S^k must be below S^l. The lexicographic ordering already ensures

S^k is not left of S^l. For all $l > k$,

$$S^l_{0x} + D(1 - Y^{k,\ell}_{\text{right}}) \geq S^k_{1x} + 1 \tag{3}$$

$$S^k_{0y} + D(1 - Y^{k,\ell}_{\text{above}}) \geq S^l_{1y} + 1 \tag{4}$$

$$S^l_{0y} + D(1 - Y^{k,\ell}_{\text{below}}) \geq S^k_{1y} + 1 \tag{5}$$

$$Y^{k,\ell}_{\text{right}} + Y^{k,\ell}_{\text{above}} + Y^{k,\ell}_{\text{below}} \geq 1, \tag{6}$$

where constant $D \geq \max(M_C + 1, N_C + 1)$ is a large constant, and 1's are added to avoid strict inequalities.

However, we still have the problem of computing sub-footprint area, which is naturally a product of two variables. Next, in Section 3.3, we introduce indicator variables for whether each chip is covered. These provide sub-footprint areas, and may render the disjunctive constraints obsolete for some objective functions.

3.3 Chip Indicator Variables

We denote the priority of chip $C^{i,j}$ as $p_{i,j}$. We introduce chip-in-sub-footprint indicator variables $\chi^k_{i,j}$ where

$$\begin{cases} \chi^k_{i,j} = & 1 \text{ if } C^{i,j} \in S^k, \\ & 0 \text{ otherwise.} \end{cases} \tag{7}$$

The *area* of a sub-footprint is the number of chips contained within it: $A^k = \sum_{i,j} \chi^k_{i,j}$.

Sub-footprints having empty intersection is equivalent to a chip being in only one sub-footprint, modeled by the constraint $\sum_k \chi^k_{i,j} \leq 1 \,\forall\, i, j$.

3.3.1 Chip in Footprint, χ

To properly set each χ we define four indicator variables V_ℓ, with $\chi = 1$ iff all V_ℓ are 1. We let $V_{\text{right}} = 1$ iff C is to the right of S_0, etc.

To ensure chips outside the sub-footprint have $V = 0$,

$$S_{0x} - (1 - V_{\text{right}}) \cdot D \leq C_x, \tag{8}$$

$$S_{0y} - (1 - V_{\text{above}}) \cdot D \leq C_y, \tag{9}$$

$$S_{1x} + (1 - V_{\text{left}}) \cdot D \geq C_x, \tag{10}$$

$$S_{1y} + (1 - V_{\text{below}}) \cdot D \geq C_y. \tag{11}$$

For the converse, chips inside have $V = 1$, by

$$S_{0x} + V_{\text{right}} \cdot D \geq C_x + 1, \tag{12}$$

$$S_{0y} + V_{\text{above}} \cdot D \geq C_y + 1, \tag{13}$$

$$S_{1x} - V_{\text{left}} \cdot D \leq C_x - 1, \tag{14}$$

$$S_{1y} - V_{\text{below}} \cdot D \leq C_y - 1. \tag{15}$$

For some variations, these four constraints are not required because the objective will ensure that optimal solutions satisfy them.

To force $\chi = 1$ if all V are 1, we add the constraint

$$\chi \geq V_{\text{right}} + V_{\text{above}} + V_{\text{left}} + V_{\text{below}} - 3. \tag{16}$$

And to ensure $\chi = 0$ if any V is 0, we constrain

$$\chi \leq V_\ell \,\forall \ell. \tag{17}$$

3.4 Area and Bandwidth Constraints

The area A^k of sub-footprint S^k is simply the number of chips within the sub-footprint, so the optimization problem defines variables A^k with constraints $A^k = \sum_{i,j} \chi^k_{i,j}$. The area is important for bandwidth considerations. The system is designed so that, given bandwidth B, each sub-footprint gets bandwidth B/K, and the bandwidth per chip is B/KA^k. Here B and K are constants. Consequently, larger area sub-footprints get smaller bandwidths per chip; We will revisit this issue in Section 3.5.

Since each chip requires a minimum bandwidth for high fidelity, the bandwidth constrains the total number of chips that can be downlinked, and the maximum area of a single sub-footprint. We constrain

$$K \cdot A^k \leq B. \tag{18}$$

3.5 Objective Function

We describe the principles behind each of the objective function terms. The primary criterion is to maximize the priority of chips covered by sub-footprints. A *reward term* captures the benefit of covering a chip with a sub-footprint:

$$R = \sum_{k \in K} \sum_{i,j} \chi^k_{i,j} \cdot p_{i,j}. \tag{19}$$

We explicitly penalize covering zero-priority chips:

$$P_0 = \gamma \sum_{k,i,j} \chi^k_{i,j}, \tag{20}$$

where γ is a constant, less than the smallest positive priority. This is necessary to ensure sub-footprints are as small as possible while still covering the same positive-priority chips, because we are constrained by the bandwidth, rather than optimizing bandwidth.

A secondary criterion is to place high-priority chips in smaller sub-footprints, because they get more bandwidth devoted to them from the fixed bandwidth per sub-footprint. This inspires a *penalty term* for a chip that increases with its value and the containing sub-footprint's area:

$$P_A = \epsilon \sum_k \sum_{i,j} A^k \cdot \chi^k_{i,j} \cdot p_{i,j}, \tag{21}$$

where ϵ is another such constant.

Given how bandwidth is assigned to chips, It appears natural to *divide* the reward by the area, but this leads to a non-linear objective function, which is more expensive to solve, so we subtract the penalty instead. Our objective function F is then

$$F = R - P_A - P_0 \tag{22}$$

The problem with this formulation is that it is still non-linear: (21) contains the product of variables $\chi^k_{i,j}$ and A^k. To keep the constraints linear, we introduce new real valued variables Z to replace $\chi \cdot A$. Consequently, the new reward term replacing (19) and (21) is

$$\tilde{R} = \sum_k \sum_{i,j} z^k_{i,j} \cdot p_{i,j}. \tag{23}$$

We design constraints so that Z is nearly equal to χ, but exactly zero in the case that $\chi = 0$, and penalized by sub-footprint area A. We let Z be real-valued between 0 and 1, and enforce $z^k_{i,j} \leq \chi^k_{i,j}$, so $\chi = 0 \implies Z = 0$.

We introduce slack variables s such that $\chi = 1 \implies s = 0$. We let s be real-valued in $[0, 1]$, with $s_{i,j} \leq 1 - \chi^k_{i,j}$. The sub-footprint area penalty is built into Z via the constraint that

$$z^k_{i,j} - s_{i,j} \leq \chi^k_{i,j} - \epsilon A^k. \tag{24}$$

And our objective function is

$$\tilde{F} = \tilde{R} - P_0. \tag{25}$$

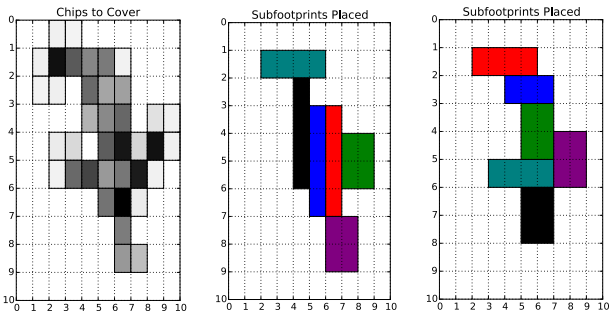
3.5.1 Optimized Bandwidth, Constrained Coverage

We also consider the problem of minimizing the required bandwidth, subject to the constraint that all chips above some threshold value must be covered. For this variant, we use the non-overlapping rectangle constraints from Section 3.2. We may either compute area using the chip indicator variables from Section 3.3, or approximate area by footprint perimeter.

4 Greedy Solution

While the optimization problem provides high quality solutions, solving it is time consuming, mostly due to the large number of binary or integer variables. E.g., order minutes for the example we demonstrate in this paper. In contrast, geometric solutions for problems of this size typically take fractions of a second. Hence, we explore a greedy heuristic for choosing sub-footprints, in order to inform the optimization problem. We iteratively choose the next sub-footprint that covers the highest priority chips, but does not overlap with any prior sub-footprint. The sub-footprint must have area $A^k \leq B/K$, but may otherwise be any shape of rectangle. We precompute the allowable sub-footprint shapes. The allowable shape with the largest area is not always chosen, e.g., 2×4 may be preferable to 3×3.

By definition, the greedy solution is not better than the optimal solution. Figure 1 displays the results of a greedy solution in Figure 1(b), and an optimization solution Figure 1(c), for the same set of chips from Figure 1(a). In this case the solutions cover different chips, and use sub-footprints of different shapes. The greedy solution appears less compact, with some small gaps of uncovered chips, as one might expect.



(a) Chips to cover. **(b)** Greedy solution. **(c)** Optimal MIP.

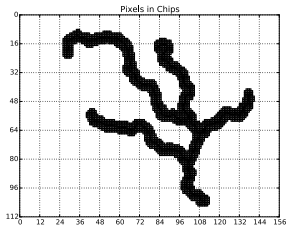
Figure 1: A greedy solution vs. an optimized solution for the same chip layout. In the left column, darker chips have higher value. This example was chosen to illustrate the potential for a large difference between the greedy and optimal solutions; in many examples they are more similar, or even identical.

5 Shifting Pixels to Find Optimal Sub-footprints

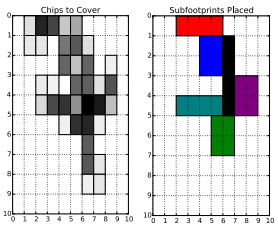
In Section 3, we considered the case that chips were in fixed position relative to the pixels of interest. Here, we explore the freedom to shift the entire footprint by a few pixels, in order to align chip boundaries with the pixels of interest, to enable higher value sub-footprint placements. Recall the priority of a chip is the sum of priorities of its pixels. Another possible benefit of shifting is to concentrate high priority pixels into fewer chips. We may shift by up to $\Delta x - 1 \times \Delta y - 1$ pixels, after which the the problem statement repeats itself through periodic symmetry.

Figure 2(a) shows an example corresponding to the Mississippi River. The image has $M_C \times N_C = 10 \times 10$ chips, $M \times N = 160 \times 120$ pixels. The darker pixels have higher priorities. We consider a maximum of $K = 6$ sub-footprints and bandwidth $B = 24$.

We exhaustively consider all possible shifts, and for each find the greedy solution. We also find the *perfect* solution of the B highest priority chips, independent of the sub-footprint geometric constraints. The perfect solution value is an upper bound on the optimal solution, just as the greedy solution is a lower bound. Evaluating all of these configurations is faster than solving a single optimization instance. The goal of this study was to confirm that shifting can provide improved solutions in practice, and further to provide guidelines for selecting which shifts are worth running the optimization algorithm over. Figure 2(b) displays outputs from two different shifts. The corresponding objective function values and generated sub-footprints are significantly different.



(a) The Mississippi River, courtesy Wikipedia. Darker pixels are more valuable.



(b) An optimal solution for the same set of pixels, but shifted to lie in different chips.

Figure 2: Shifting the pixels to align with chip boundaries results in a qualitatively different sub-footprint layout, and the value of the solution varies by about 3%, compared to Figure 1.

5.1 Selecting Layouts for MIP

The MIP runtime is too expensive to run over all shifts. Instead we seek a subset of promising shifts, and run the MIP only on those. We use the following variant of k-means clustering, where k is the budget of the number of MIPs we can run. The first cluster center is the layout with highest greedy value. Each subsequent cluster, up to k, is the layout with maximum *distance* from all prior clusters. We then form clusters by assigning all layouts to their closest center. We choose the layout with the highest greedy value in a cluster as its *exemplar*. We then run the MIP the exemplars. Figure 4 shows our example with $k = 10$. Figure 3 provides evidence that running the MIP on the exemplar is sufficient within a cluster, but that it is worthwhile running the MIP on multiple clusters.

We define the *distance* D between two layouts as a kind of transport or Wasserstein distance, the minimum *work* needed to transform one layout A to another B [8]. *Work* is defined as follows. The work for translating the entirety of A by an integral number of chips is zero. Each sub-footprint in A is matched to a sub-footprint in B; the work of re-labeling sub-footprints to provide a different matching is zero. The work of translating a chip from a sub-footprint in A to a sub-footprint in B is the L_1 distance between them.

This definition captures both the differences in the covered chips, but also the differences in how those chips are covered. (We also experimented with selecting layouts based on the *perfect* solution, but this was not as predictive. Specifically, sometimes the optimal solution was low. See Figure 3.)

Computing D is expensive, and getting it exact is unimportant, so we approximate it by D'. (We must compute Kn distances, where n is the number of layouts, and solving D is an optimization problem with many discrete decisions.) The D' distance between two sub-footprints is the sum of the L_1 distances between a greedy min L_1 pairing of their chips. The D' distance between two layouts is the sum of the D' distances between a greedy pairing of their sub-footprints. For the D' distance between layouts, we consider four translations of A, by up to one chip, taking advantage of the knowledge of how pixels were shifted.

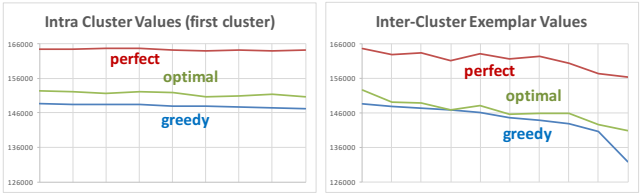


Figure 3: Intra- and inter-cluster variability. Note the strong correlation between the greedy and optimal solution within a cluster; it suffices to run the MIP on the layout with the best greedy value within a cluster. The correlation is less strong between clusters, because of the qualitatively different geometric differences between the greedy and optimal solutions from one cluster to another. For example, for the 4th cluster, the greedy solution is optimal, while for other layouts there are significant gaps. Hence it is worthwhile running the MIP on the exemplar from each cluster.

6 GeoPlace Open Source Software

We provide optimization and heuristic methods for sub-footprint placement in the open-source software GeoPlace [1]. We provide several test models. The repository also holds optimization formulations for the related problem of placing a mosaic of footprints to cover a large area, and for scheduling the placement of footprints.

Optimization models were expressed in the Pyomo[2] [4] modeling language. The development environment was Linux and the source code was primarily developed in Python 2.7. Pyomo uses TPL's for the actual solvers, we primarily used CPLEX[3] (commercial) and Gurobi[4] (free academic use license). The repository includes additional Python and C++ routines for visualizing solutions.

7 Conclusions

We have shown optimization based-solutions to a rectangle placement problem. We have used greedy heuristics and geometric clustering to select promising and

[1] GeoPlace `https://github.com/cgvalic/GeoPlace`
[2] Pyomo `http://www.pyomo.org`
[3] CPLEX `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer`
[4] Gurobi `http://www.gurobi.com`

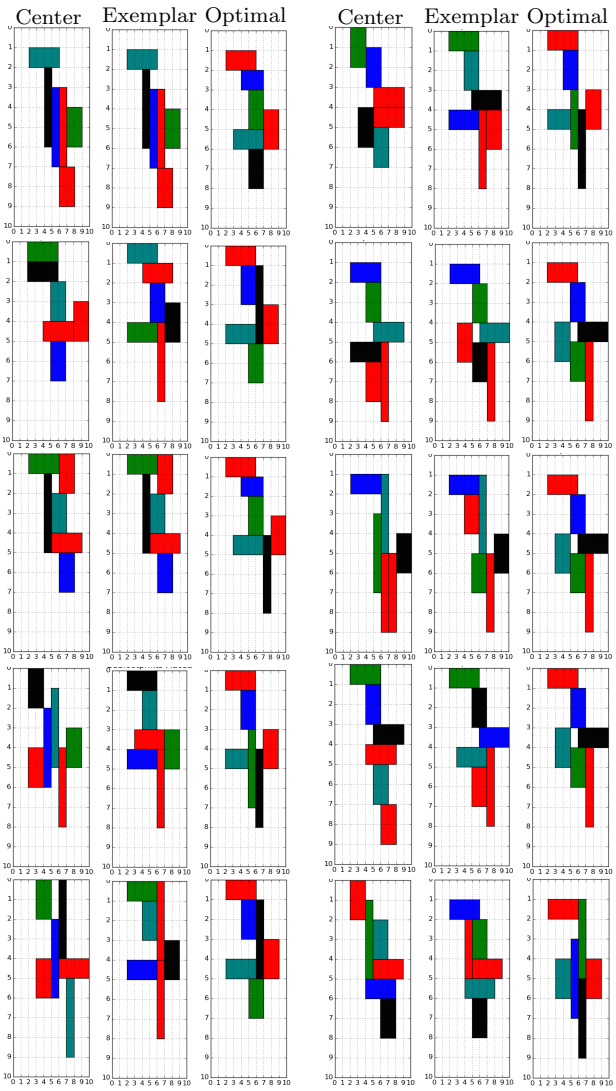


Figure 4: Greedy solutions for cluster centers and exemplars, and MIP optimal solution for exemplars. We consider 181 patterns of shifting up to 12 × 16 pixels. Here cluster centers (exemplars) = { 165(165), 38(114), 86(115), 176(173), 108(108), 188(189), 48(81), 85(142), 45(99) 56(23)}.

geometrically-distinct problem instances on which to invest the time to run the optimization solution. The problems are small enough that the complexity of the geometric algorithms is not an issue. Moreover, they are orders of magnitude faster than the optimization-based approaches. While the optimization problem is expensive to solve, it offers some advantages in terms of flexibility and utility. The approach is extensible to new objectives and constraints. The optimization problem may be incorporated into larger ones, such as scheduling. In the broader project, we developed scheduling software, and footprint and subfootprint placement algorithms.

Here we have sought to optimize the covered chips subject to a bandwidth constraint. In future work, it would be worth considering Pareto optimal solutions to the multi-objective optimization problem of maximizing chip coverage while minimizing bandwidth. We hope the community is inspired to develop geometric algorithms for more satellite planning and tasking problems.

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), Applied Mathematics Program. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and 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-NA-0003525.

References

[1] M. Andersson, J. Gudmundsson, and C. Levcopoulos. Chips on wafers, or packing rectangles into grids. *Computational Geometry*, 30(2):95 – 111, 2005.

[2] P. Bose, M. van Kreveld, A. Maheshwari, P. Morin, and J. Morrison. Translating a regular grid over a point set. *Computational Geometry*, 25(1):21 – 34, 2003.

[3] J. Frank, A. Jónsson, R. Morris, and D. E. Smith. Planning and scheduling for fleets of Earth observing satellites. In *Proceedings of Sixth Int. Symp. on Artificial Intelligence, Robotics, Automation & Space*, 2001.

[4] W. E. Hart, C. Laird, J.-P. Watson, and D. L. Woodruff. *Pyomo–optimization modeling in Python*, volume 67. Springer Science & Business Media, 2012.

[5] 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)*, pages 547–552, June 2011.

[6] D. Song, A. F. van der Stappen, and K. Goldberg. An exact algorithm optimizing coverage-resolution for automated satellite frame selection. In *Int. Conf. on Robotics and Automation*, volume 1, pages 63–70, April 2004.

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

[8] R. C. Veltkamp. Shape matching: Similarity measures and algorithms. In *Proceedings Int. Conf. on Shape Modeling and Applications*, pages 188–197, May 2001.

[9] F. Xhafa, J. Sun, A. Barolli, A. Biberaj, and L. Barolli. Genetic algorithms for satellite scheduling problems. *Mobile Information Systems*, 8(4):351–377, Oct 2012.