ESSAYS ON INTEGER PROGRAMMING IN MILITARY AND POWER MANAGEMENT APPLICATIONS

by

Serdar Karademir

BSc, Middle East Technical University, 2006

MSc, Middle East Technical University, 2008

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2013

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Serdar Karademir

It was defended on

July 1st, 2013

and approved by

Oleg A. Prokopyev, Ph.D., Associate Professor, Department of Industrial Engineering

Nan Kong, Ph.D., Assistant Professor , Biomedical Engineering, Purdue University

Jayant Rajgopal, Ph.D., Associate Professor, Department of Industrial Engineering

Andrew J. Schaefer, Ph.D., Professor, Department of Industrial Engineering

Dissertation Director: Oleg A. Prokopyev, Ph.D., Associate Professor, Department of Industrial Engineering

ESSAYS ON INTEGER PROGRAMMING IN MILITARY AND POWER MANAGEMENT APPLICATIONS

Serdar Karademir, PhD

University of Pittsburgh, 2013

This dissertation presents three essays on important problems motivated by military and power management applications. The array antenna design problem deals with optimal arrangements of substructures called subarrays. The considered class of the stochastic assignment problem addresses uncertainty of assignment weights over time. The well-studied deterministic counterpart of the problem has many applications including some classes of the weapon-target assignment. The speed scaling problem is of minimizing energy consumption of parallel processors in a data warehouse environment. We study each problem to discover its underlying structure and formulate tailored mathematical models. Exact, approximate, and heuristic solution approaches employing advanced optimization techniques are proposed. They are validated through simulations and their superiority is demonstrated through extensive computational experiments. Novelty of the developed methods and their methodological contribution to the field of Operations Research is discussed through out the dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

- to my parents

This dissertation is dedicated to my parents *Ziya* and *Ayten*, my sister *Naciye*, and my brother *Serhat*, for their unconditional love and support. *Nermin*, only you know the sacrifices that have been made to be where I stand today; no words could express my feelings.

I would like to thank my dear friends who have been there for me when I needed. I appreciate and thank for the academic support of my advisor. In my heart, I am in debt for anyone who has contributed to this dissertation in one way or another. Finally, to me, this dissertation exists because it concludes many things.

"It's important in life to conclude things properly. Only then can you let go. Otherwise you are left with words you should have said but never did, and your heart is heavy with remorse."

-Life of Pi

1.0 INTRODUCTION

This dissertation covers three important problems motivated by military and power management applications. We study each problem to discover its underlying structure and formulate mathematical models that capture details specific to the problem's context. Using advanced optimization techniques, we develop specialized exact, approximate, and heuristic solution methods that are applicable in real-life scenarios. We validate our approaches and demonstrate their superiority through extensive computational experiments and simulations. This chapter provides a brief introduction to each problem; we give a problem statement, an overview of the related literature, and a summary of our contributions.

1.1 PHASED ARRAY ANTENNA DESIGN

Chapter 2 of this dissertation explores a problem arising in array antenna design. An *antenna* is a device that can rapidly scan objects in space by emitting a well-formed beam (e.g., radio signal) in the direction of objects and then listening to the reflection of the beam from objects. A phased array antenna is composed of many stationary antenna elements. Phased array antennas are a crucial part of modern infrastructure with a wide range of applications in defense, communication and surveillance. Details on theoretical foundations of antenna designs and reviews of the current antenna technology are provided by [32, 88, 98].

The main beam of a modern phased array antenna is electronically steered. Electronic control at the antenna element level is ideal, but a large number of elements in a typical phased array antenna results in a complex engineering problem, whose solution is often too expensive to implement [90]. Therefore, a group of elements is used to form a *subarray*,

which is treated as an oversized element, and the control input is introduced at the subarray level. However, identical rectangular subarrays—as used in practice—introduce *periodicity* that creates *quantization sidelobes*. Simply speaking, a sidelobe is a beam (typically of a smaller magnitude) with a direction different from the main beam direction of the antenna. Such undesired radiation greatly reduces the quality of the pattern generated by antennas, and in modern electronic warfare, antennas can be deceived using *sidelobe jamming*. One approach recently proposed for reducing quantization sidelobes is based on *irregular* tilings with *polyomino*-shaped subarrays [89, 90].

A polyomino is a generalization of the *domino* and is created by connecting a fixed number of unit squares along edges [29, 61]. The number of different polyominoes, excluding rotations and reflections, grows exponentially fast; 2, 108, and 63,600 different polyominoes can be instructed using 3, 7, and 12 unit squares, respectively. The vast majority of the related literature is focused on polyomino enumeration [18, 42, 47, 54, 76] and classification [11, 21, 36, 59, 60].

Our work is focused on tiling a rectangular region using a given set of polyominoes. Furthermore, among all possible tilings, we are interested in irregular ones. To the best of our knowledge, this problem has not been studied by the operations research community in the past. We develop a set-theoretic description of *irregular polyomino tiling problem* and formulate it as a nonlinear exact set covering model, where irregularity of a tiling is measured using the information-theoretic *entropy* concept. We develop an advanced exact solution method based on the branch-and-price framework along with novel branching and bounding ideas. To solve large-size instances, we propose efficient heuristics that are shown to provide approximation guarantees. We report encouraging computational results including actual phased array antenna simulations that demonstrate significant reductions in peak sidelobes for irregular polyomino tilings obtained using the developed approach.

1.2 STOCHASTIC ASSIGNMENT

The class of stochastic assignment problem studied in Chapter 3 is a two-stage stochastic extension of the classical deterministic linear assignment problem (LAP). The well-known LAP consists of assigning a set of jobs to a set of agents such that the total weight of the assignment is maximized. There exists many extensions of LAP with a wide range of applications including the weapon-target assignment [33, 94, 99]. In our problem setting, assignment weights are subject to change over time. Decision maker has to partially assign jobs to agents in the first stage. Then, based on the scenario realized, the assignment is completed in the second stage.

There has been a rapid increase in applications of stochastic optimization methodology to a variety of important real-world problems including resource allocation, planning, logistics, scheduling, and health care [24, 102]. A number of studies has been focused on the design of exact solution methodologies, with typical examples in [1, 2, 7, 95, 105, 108]. Unfortunately, many standard deterministic optimization problems that are solvable in polynomial time, become NP-hard if considered in the stochastic environment, e.g., maximum weight matching [81] and spanning tree [52] problems. Thus, we can not expect to be able to solve general stochastic integer and combinatorial optimization problems exactly for large input sizes, and similar to the deterministic optimization literature, it is desirable to design polynomial time approximation algorithms with reasonable performance guarantees. Recent examples of this type of work include methods for solving two-stage stochastic extensions of the shortest path [62], minimum spanning tree [46], min-cut [45, 62], and Steiner tree [66] problems. For more detailed introductions and surveys to stochastic programming, we refer the reader to [24, 96, 102, 104, 107, 110].

Kong and Schaefer [81] consider the two-stage stochastic maximum weight matching problem on general graphs. They show that the problem is NP-hard and propose a greedy $\frac{1}{2}$-approximation algorithm. Escoffier et al. [50] prove that the two-stage stochastic maximum weight matching problem is APX-complete even for bipartite graphs of maximum degree 4 and general graphs of degree 3. Based on the concepts from [81], they also provide a

greedy $\max\{\frac{K}{2K-1}, \frac{\Delta}{2\Delta-1}\}$-approximation algorithm, where K is the number of scenarios in the second-stage and Δ is the degree of the bipartite graph.

The work covered in Chapter 3 is essentially built on these two studies [50, 81]. Since the maximum weight matching problem on bipartite graphs can be polynomially reduced to the linear assignment problem, the two stage stochastic linear assignment problem is also APX-complete. We propose a necessary optimality condition that generalizes and unifies the key ideas behind the two algorithms by Kong and Schaefer [81] and Escoffier et al. [50]. Then based on this optimality condition, we design a new greedy approximation algorithm. This algorithm is computationally efficient as it employs Hungarian Method [83], the most popular approach to solve the linear assignment problem. While the developed approach preserves the existing approximation guarantees, we are not able to prove whether it provides a strictly better approximation bound. However, analytical observations and extensive computational results indicate that the proposed algorithm has strictly better performance on some rather broad classes of the two-stage stochastic linear assignment problems.

1.3 SPEED SCALING OF PARALLEL PROCESSORS

Chapter 4 of the dissertation is concerned with a class of nonlinear mixed integer knapsack problems motivated by the speed scaling of heterogeneous parallel processors. This is an active research area in the computer science literature following the technological shift to multi-core processors, increased online transaction volumes handled by servers with hundreds of processors, and establishment of data processing centers with thousands of servers. In this environment, electricity becomes the major cost item and optimal processor scheduling that minimizes the total energy consumption is a crucial issue.

Dynamic speed scaling literature can be traced back to [116]. Majority of the related studies is focused on the single processor case. Jobs with deadlines are considered by [14, 16, 68, 75, 116]. When deadlines do not exist, the aim has been to optimize some scheduling metric in addition to minimizing energy consumption [5, 15, 17, 38, 85, 84].

The multiprocessor case has not attracted much attention, primarily due to its complexity. Nevertheless, heterogeneous processors are becoming more common and require special treatment [28]. Similar to the single processor case, work in [4, 6, 23, 86, 64, 65] emphasizes the scheduling aspect of speed scaling. A more thorough review is provided in [3].

The focus of our work is a relatively high-level problem setting that includes heterogeneous parallel processors with sleep states. Given a processing requirement, our goal is to select a subset of processors and distribute the load over these processors so as to minimize the total energy consumption. The energy consumption curves are convex functions and we refer to the problem as *speed scaling with convex power functions* (SSCPF). SSCPF is naturally modeled as a mixed integer nonlinear program (MINLP), which can be seen as the nonlinear extension of the linear fixed charge models. Two decisions are to be made: which processors should be turned on and what should be their speeds. The latter decision is known as the continuous nonlinear resource allocation problem (NRAP) [73]. Most of the algorithms devised for NRAP are based on the Karush-Kuhn-Tucker (KKT) optimality conditions and use of ϵ-accuracy notion [30, 31, 71, 72].

Using KKT conditions, first, we identify several polynomially solvable cases of SSCPF and show that its general case is NP-hard. Next, a pseudo-polynomial dynamic programming algorithm for a special case of SSCPF is presented. We also show that a simple greedy heuristic provides an n-factor approximation guarantee for SSCPF. Using this approximation algorithm, the proposed dynamic program is converted into a fully polynomial-time approximation scheme (FPTAS).

For the general case of our problem, we also implement the *outer approximation* algorithm [48, 53, 100]. The underlying motive for outer approximation is to relax the MINLP into a mixed integer linear program (MILP) through approximation of the nonlinear objective at certain points. The relaxation is iteratively refined using additional cuts within a branch-and-bound framework until an optimal solution of the original problem is found.

Finally, we carry out comprehensive computational experiments aiming at comparing the naive approach of using an off-the-shelf mixed integer programming solver, the outer approximation implementation, and the dynamic programming method. Our results clearly demonstrate the superiority of developed approaches.

1.4 OVERVIEW OF THE DISSERTATION

The remainder of the dissertation is organized as follows. Chapter 2 details our work on design of phased array antennas. Stochastic assignment problem is discussed in Chapter 3. Finally, we present our results for speed scaling of multiple heterogeneous processors in Chapter 4. Chapter 5 concludes this dissertation.

2.0 IRREGULAR POLYOMINO TILING VIA INTEGER PROGRAMMING WITH APPLICATION IN PHASED ARRAY ANTENNA DESIGN

2.1 INTRODUCTION

A *polyomino* is a generalization of the *domino* and is created by connecting a fixed number of unit squares along edges [29, 61]. Figure 1 illustrates the first five families of polyominoes $\mathcal{F}_\ell$, $\ell = 1, \ldots, 5$. The number of different polyominoes, excluding rotations and reflections, grows exponentially fast in ℓ [61]. There are 2, 108, and 63,600 different polyominoes for $\ell = 3$, 7, and 12, respectively. The vast majority of the related literature is focused on polyomino enumeration [18, 42, 47, 54, 76] and classification [11, 21, 36, 59, 60]. Tiling finite, general regions of the plane is known to be NP-complete even when using only the right-tromino [93]. This result holds under the assumption that there is no restriction on the number of copies of each polyomino used. If such restrictions exist (e.g., solving a puzzle), then tiling a finite, rectangular, simply connected region is NP-complete if the pieces have at least polylogarithmic area in the dimensions of the tiling region [43].

In this chapter, we focus on tiling a finite, rectangular, simply connected region using a given finite set of polyominoes, without any restriction on the number of each polyomino copies used. This problem is NP-complete [22, 55]. Another direct proof of this fact can be provided using reduction from the classical NP-complete *integer knapsack feasibility problem* [97]. Specifically, given nonnegative integers $s_1 < s_2 < \ldots < s_n$, and K, we need to check whether there exist nonnegative integers x_1, x_2, ..., x_n that satisfy

$$s_1 x_1 + s_2 x_2 + \ldots + s_n x_n = K. \tag{2.1}$$

Consider a set $\mathcal{P}$ of rectangular polyominoes $1 \times s_1, \ldots, 1 \times s_n$. Then it is easy to observe that a rectangular $1 \times K$ strip can be exactly covered (tiled) by some polyominoes from $\mathcal{P}$ iff (2.1) has an integral solution. Further detailed discussion on polyomino tilings can be found in works by [8, 25, 59, 60, 91, 101] and references therein.

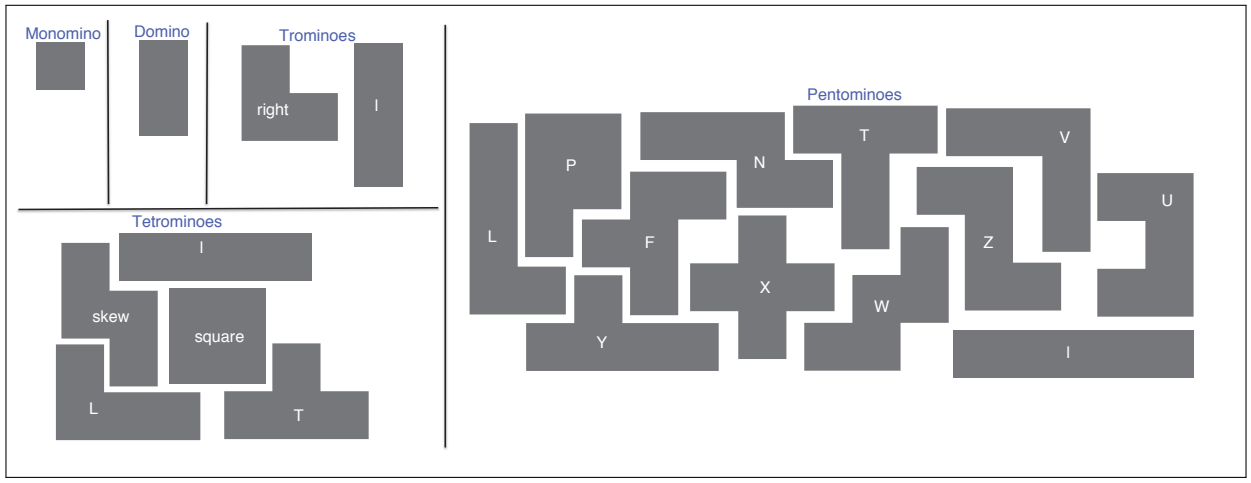


Figure 1: Monomino ($\mathcal{F}_1$), domino ($\mathcal{F}_2$), tromino ($\mathcal{F}_3$), tetromino ($\mathcal{F}_4$), and pentomino ($\mathcal{F}_5$) families.

Our work is motivated by a recent application of polyomino tilings in the design of *phased array antennas* [89, 90]. An antenna is a device that can rapidly scan objects in space by emitting a well-formed beam, most of the time a radio signal, in the direction of the objects and then listening to the reflection of the beam from objects. A phased array antenna is composed of many stationary antenna elements.

Phase shift and *time delay* are the key concepts behind electronically steering the beam in modern phased array antennas. Electronic steering allows for instantaneous positioning, multiple object tracking, and simultaneous precision tracking and background mapping. This technology replaces mechanically steered antenna designs, where beam direction is constant and the antenna itself is mechanically positioned in space. Moreover, mechanically steered antennas are more vulnerable to electronic countermeasures as their beam position can be predicted. Phased array antennas are a crucial part of modern infrastructure with a wide range of applications in defense, communication, astronomy, meteorology, and surveillance.

Details on theoretical foundations of antenna designs and reviews of the current antenna technology are provided by [32, 88, 98].

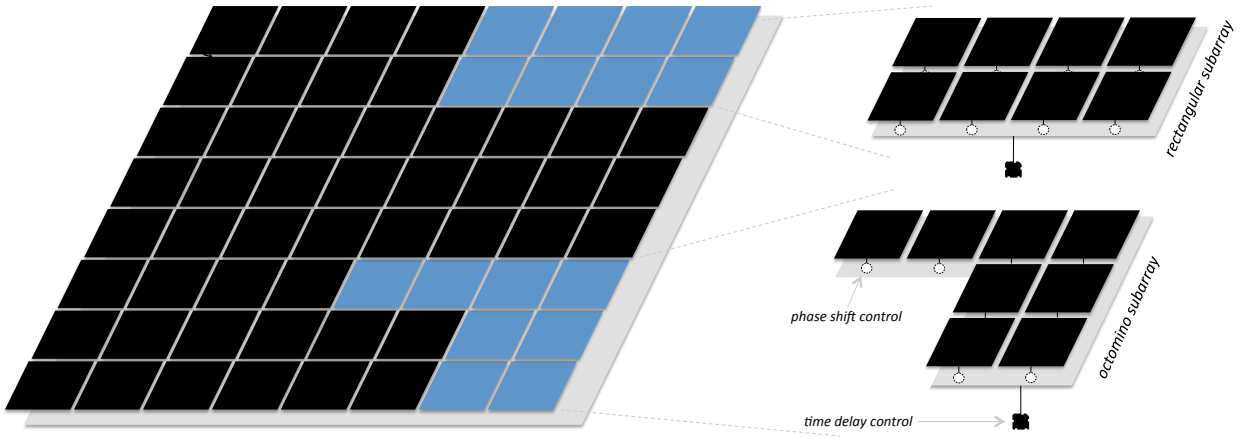
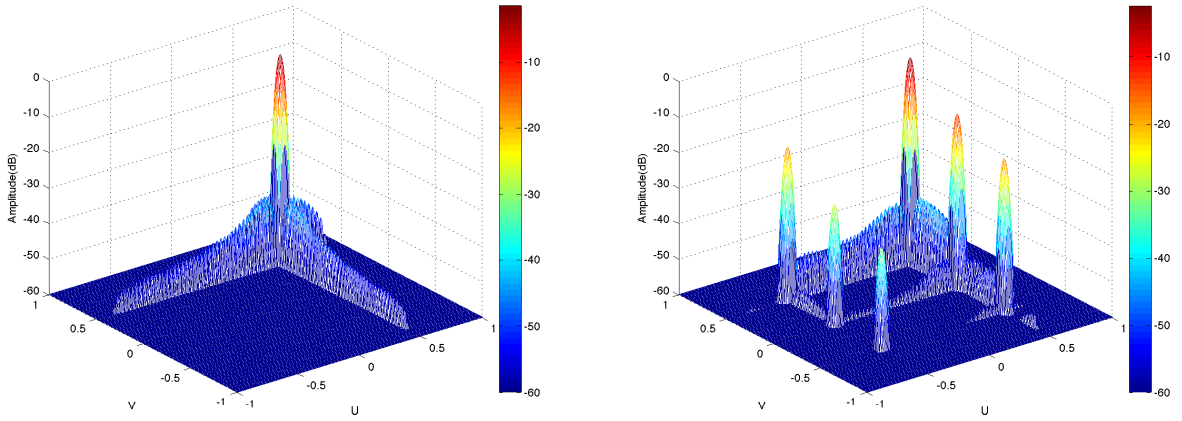


Figure 2: Array and two subarray examples: rectangular and L-shaped.

Controls (phase shift and time delay) at the antenna element level is ideal, but, a large number of elements (possibly, tens of thousands) in a typical phased array antenna results in a complex engineering problem, whose solution is often too expensive to implement [90]. Therefore, a group of elements is used to form a *subarray*, which is treated as an oversized element, and the control input (e.g., time delay) is introduced at the subarray level. An example of rectangular subarray with time delay control at the subarray level and phase shift control at the element level is given in Figure 2.

However, identical rectangular subarrays—as used in practice—introduce *periodicity* that creates *quantization sidelobes*. Simply speaking, a sidelobe is a beam (typically of a smaller magnitude) with a direction different from the main beam direction of the antenna. Such undesired radiation greatly reduces the quality of the pattern generated by antennas, and in modern electronic warfare, antennas can be deceived using *sidelobe jamming*. Figure 3a shows a well-formed beam for a 64×64 array with element level time delay control. Five significant sidelobes present in Figure 3b are caused by periodically tiled 2×4 rectangular subarrays with subarray level time delay control.

One approach recently proposed for reducing quantization sidelobes is based on *irregular* tilings with polyomino-shaped subarrays [89, 90]. As modern phased array antennas



(a) Radiation pattern for 64×64 array with time delay control at element level.

(b) Radiation pattern for 64×64 array with time delay control at 2×4 rectangular subarray level.

Figure 3: Radiation patterns for different time delay control scenarios.

contain thousands of antenna elements, this experimental observation poses a large-scale combinatorial optimization problem. Specifically, given a set of polyominoes (i.e., possible subarray shapes), the question is how to construct an irregular exact (or almost exact) tiling of a given rectangular region (i.e., phased array antenna). Examples of rectangular and L-shaped subarray structures are illustrated in Figure 2.

The rest of this chapter is organized as follows. In Section 2.2, we formulate the *irregular polyomino tiling problem* as a nonlinear exact set covering model, where irregularity of a tiling is incorporated into the objective function using the information-theoretic entropy concept. While the proposed model can be linearized, standard solvers cannot handle even small-size instances of the problem. Thus, an advanced exact solution method based on the branch-and-price (B&P) framework along with novel branching and bounding ideas is proposed in Section 2.3. This method is capable of solving small- and medium-size problem instances. To solve large-size instances, we propose efficient heuristics (Section 2.4) that can be shown to provide approximation guarantees under some conditions (Section 2.5). Finally, Sections 2.6 and 2.7 summarize encouraging computational results including actual

phased array antenna simulations that demonstrate significant reductions in peak sidelobes for irregular polyomino tilings obtained using the developed approach.

2.2 MODEL FORMULATION

2.2.1 Nonlinear Exact Set Covering Model

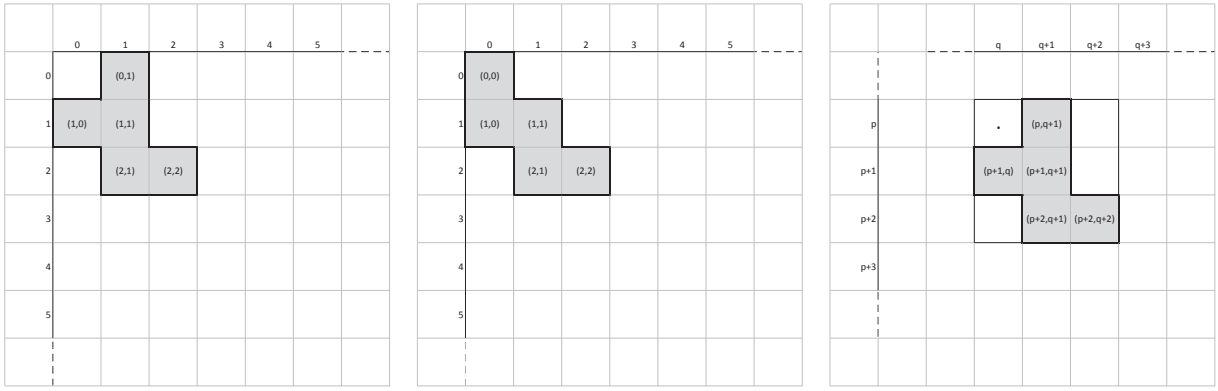
In this section we provide a set-theoretic interpretation of the problem and formulate it as a nonlinear binary mathematical program. Let $\mathcal{F}_\ell$ be the family of polyominoes with $\ell \in \mathbb{Z}_{++}$ squares. Assume that $\mathcal{F}_\ell$ is extended to include all possible rotations and reflections of its elements. The *rectangle hull* of a polyomino is the smallest rectangular box that contains the polyomino. Any $f \in \mathcal{F}_\ell$ can be described as a set of squares covered when the northwest corner of its rectangle hull is located at $(0,0)$, i.e., $f = \{(i_1, j_1), (i_2, j_2), \ldots, (i_\ell, j_\ell)\}$. Figures 4a and 4b illustrate this representation for F pentomino (reflected and rotated 180^0 clockwise) and W pentomino (rotated 90^0 clockwise).

The rectangular $(m \times n)$ region to be tiled is described as the set $\mathcal{B} = \{(i,j) : 0 \leq i < m;\, 0 \leq j < n;\, i,j \in \mathbb{Z}\}$. Henceforth, we refer to $\mathcal{B}$ as a *board*. Denote by $\mathcal{P} = \{f^1, f^2, \ldots, f^K\} \subseteq \mathcal{F}_\ell$ a subset of polyominoes that can be used for tiling $\mathcal{B}$. For each $f^k \in \mathcal{P}$ define $f^k_{pq} = \{(i+p, j+q) : (i,j) \in f^k\}$ as the set of squares covered when the northwest corner of the rectangle hull of f^k is located at (p,q) (see Figure 4c), i.e., f^k_{pq} is simply a translation of f^k. Thus, the tiling problem $(\mathcal{B}, \mathcal{P})$ can be defined as follows: given a board $\mathcal{B}$ and a set of polyominoes $\mathcal{P}$, find an exact cover of $\mathcal{B}$ using subsets $\{f^k_{pq} : f^k_{pq} \subseteq \mathcal{B};\, f^k \in \mathcal{P}\}$.

Define $I_{ij} = \{(k,p,q) \,|\, (i,j) \in f^k_{pq}\}$ to be the set of all feasible triplets (k,p,q) (i.e., $f^k_{pq} \subseteq \mathcal{B}$) that cover square $(i,j) \in \mathcal{B}$. Finally, introducing a binary variable x^k_{pq} for each set f^k_{pq}, the exact set covering constraints can be formulated as

$$\sum_{(kpq) \in I_{ij}} x^k_{pq} \;=\; 1 \qquad\qquad \forall\, (i,j) \in \mathcal{B}\,. \tag{2.2}$$

Any feasible solution of (2.2) defines a tiling $\mathcal{T}$ for $\mathcal{B}$, where triplet (k,p,q) is used iff $x^k_{pq} = 1$. See [9, 12, 13, 57] for a detailed treatment of the set partitioning problem.



(a) $f^1 = \{_{(0,1),(1,0),(1,1),(2,1),(2,2)}\}$ (b) $f^2 = \{_{(0,0),(1,0),(1,1),(2,1),(2,2)}\}$ (c) $f^1_{pq} = \{_{(p,q+1),\ldots,(p+2,q+2)}\}$

Figure 4: Two polyominoes from pentomino family, $\mathcal{F}_5$. Polyomino f^1_{pq} is shown with its rectangle hull.

To capture the "irregularity" of polyomino tilings, we apply the *information-theoretic entropy* concept [106], formally defined as follows. Let X be a discrete random variable with finite support, where outcome i has probability p_i. Then the information-theoretic entropy of X, $H(X)$, is given by:

$$H(X) = -\sum_i p_i \lg(p_i) \ , \tag{2.3}$$

where $\lg(\cdot) \equiv \log_2(\cdot)$. When $p_i = 0$, the value of the term $p_i \lg p_i$ is assumed to be zero, which is consistent with $\lim_{p \to 0^+} p \lg(p) = 0$. Note that $H(X)$ is nonnegative and concave in p_i. Its maximum is achieved for the uniform distribution, i.e., $p_i = \frac{1}{N}$, $i = 1, \ldots, N$, as the solution to $\max\{-\sum_i p_i \lg(p_i) \ : \ \sum_i p_i = 1, \ p_i \geq 0\}$. Entropy measures the uncertainty inherent in a probability distribution. Also, information-theoretic entropy is closely related to the notion of entropy in statistical mechanics. Specifically, consider each outcome of X to be a *micro-state* that some physical system can possibly occupy. The less we are certain about the micro-state of the system, the larger is its entropy. Conversely, consider a random variable with a single outcome with probability 1. Then complete knowledge about the state of the corresponding system is available; hence, its entropy is 0 (i.e., $1 \cdot \lg(1) = 0$).

Information-theoretic entropy has a number of interesting applications in various disciplines [51, 77]. It has been used to obtain statistical distributions, laws of thermodynamics, population density distributions in economics, and traffic density distributions in regional and urban planning; the goal is to choose a probability distribution that satisfies the moment constraints and maximizes entropy. The key requirement for application of the entropy concept is to define a proper probability distribution over the structural components of the problem. An interesting example is the *graph entropy*, where p_i is introduced for every vertex i as a function of certain well-structured graph-theoretic quantities, see details in [41, 109] and [40]. The maximum entropy sampling problem studied in [80, 87] is to select a subset of correlated random variables that provides maximum information. Information content of the selected subset is measured via the logarithm of the determinant of the principal minor of the covariance matrix. For the minimum entropy set covering problem studied by [35, 67], a probability distribution is defined on the cardinality of the subsets used in a cover. The aim is to find a cover that corresponds to a probability distribution with minimum entropy. This problem is a variation of the minimum cardinality set covering problem and is known to be NP-hard [67].

In our setting, we consider a square to be a unit mass and define the *center of gravity* (or the *center of mass*) for each polyomino f. Then the square that contains the center of gravity of f is denoted by $g = (i, j)$. Hence, the center of gravity for f_{pq} is given by $g_{pq} = (i + p, j + q)$. Our definition is flexible as any square within the rectangle hull of f can be designated to be its center of gravity. For example, in the phased array antenna application, the center of gravity may correspond to the square (i.e., antenna element) where the subarray's main beam is formed.

Our primary intuition is that, in a *regular* tiling with a periodic pattern (e.g., using identical rectangular subarrays), centers of gravity are expected to accumulate on certain rows and columns of $\mathcal{B}$. Specifically, let r_i and c_j denote the numbers of centers of gravity on row i and column j, respectively, for a given tiling $\mathcal{T}$. Then define the following probability distribution on the rows and columns: row i has probability $\frac{r_i}{2T}$ and column j has probability $\frac{c_j}{2T}$, where T is the total number of polyominoes used in $\mathcal{T}$. These values can be interpreted as an approximation of true probability that a particular row or column contains a center of

gravity. The periodicity in regular tilings would drift the probability distribution away from being uniform. Hence, the lower the entropy of a tiling $\mathcal{T}$ is, the more periodic $\mathcal{T}$ must be, and vice versa.

Figure 5 stands as a proof of concept to the above argument. Observe that centers of gravity are perfectly aligned when entropy is minimized, while the maximization case ensures that centers of gravity are almost uniformly distributed and the obtained tiling is noticeably irregular. This metric also provides a theoretical upper bound on the objective function value. Recall that uniform distribution has the maximum entropy, and thus,

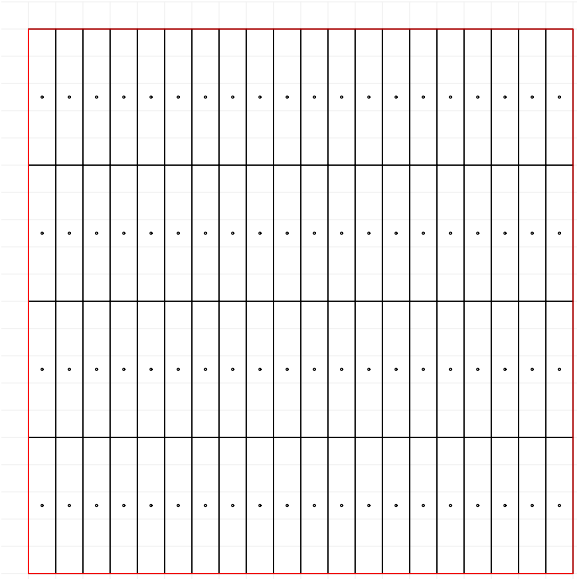
$$H(X) \leq H(\text{Uniform}) = -\sum_{i=1}^{m+n} \frac{1}{m+n} \lg\left(\frac{1}{m+n}\right) = -\lg\left(\frac{1}{m+n}\right) = \lg(m+n)\,.$$

This upper bound is important because it allows performance analysis of the heuristic algorithms discussed in Section 2.4. Note that for the tiling in Figure 5, $\lg(20+20) = 5.3219$, implying the obtained solution is at most 10^{-2} away from optimal.

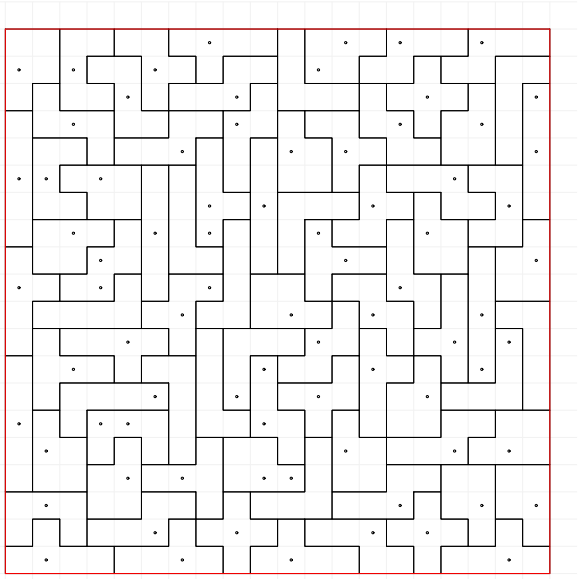
Let $R_i = \{(k,p,q)\,|\,\exists\, j \text{ such that } (i,j) = g^k_{pq}\}$ and $C_j = \{(k,p,q)\,|\,\exists\, i \text{ such that } (i,j) = g^k_{pq}\}$ be the sets of triplets with centers of gravity on row i and column j, respectively. The nonlinear mixed integer programming (MIP) formulation for the irregular *perfect* tiling problem is given by:

$$\begin{aligned}
\text{P}_{\text{NL}}: \qquad \max \quad & -\sum_i \frac{r_i}{2T} \lg\left(\frac{r_i}{2T}\right) - \sum_j \frac{c_j}{2T} \lg\left(\frac{c_j}{2T}\right) && && (2.4)\\
\text{s.to} \quad & \sum_{(kpq)\in I_{ij}} x^k_{pq} \;=\; 1 && \forall\ (i,j) \in \mathcal{B} && (2.5)\\
& r_i = \sum_{(kpq)\in R_i} x^k_{pq} && \forall\ i && (2.6)\\
& c_j = \sum_{(kpq)\in C_j} x^k_{pq} && \forall\ j && (2.7)\\
& x^k_{pq} \in \{0,1\},\ r_i,\ c_j \geq 0 && \forall\ i,j,p,q,k && (2.8)
\end{aligned}$$

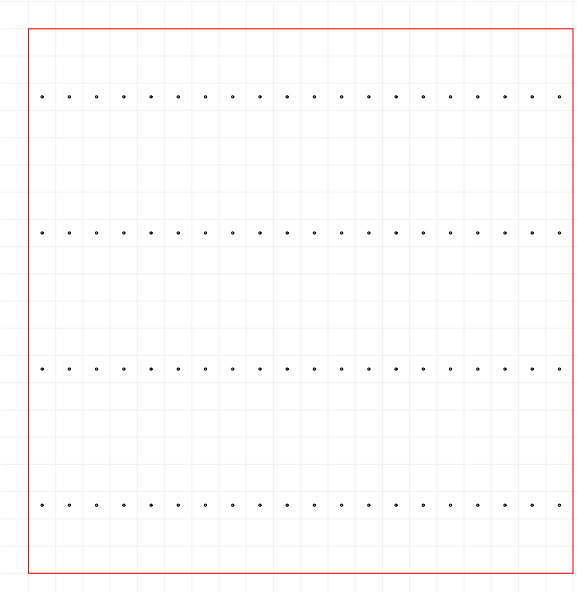
A tiling $\mathcal{T}_{\mathcal{B}}$ is called *imperfect* if it does not cover some of the squares of $\mathcal{B}$. A practically interesting extension of the tiling problem arises when only a subset of rows and columns



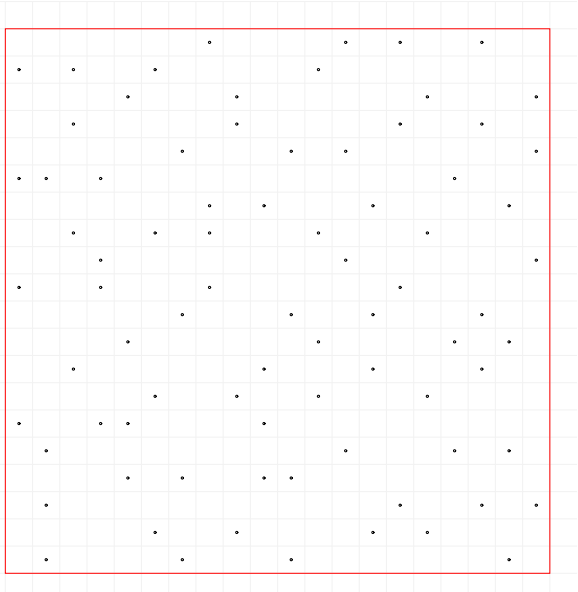
(a) Minimum entropy tiling.



(b) Maximum entropy tiling.



(c) $H(X) = 4.1610$.



(d) $H(X) = 5.3174$.

Figure 5: Minimizing versus maximizing entropy on a 20×20 board using pentomino family.

along the edges of of $\mathcal{B}$ are not required to be tiled exactly [90]. For imperfect tilings, we distinguish between the *board frame*, $\mathcal{B}_f$, and the *board center*, $\mathcal{B}_c$.

To construct an $m \times n$ imperfect tiling, consider a board $\mathcal{B} = \{(i,j) : i = 0,\ldots,m+2a-1 \text{ and } j = 0,\ldots,n+2b-1\}$, where (a,b) is the dimension of the minimum rectangle that contains each polyomino in $\mathcal{P}$. For imperfect tilings, the board $\mathcal{B}$ is partitioned into $\mathcal{B}_c = \{(i,j) : i = a,\ldots,m+a-1 \text{ and } j = b,\ldots,n+b-1\}$, the $m \times n$ board center, and $\mathcal{B}_f = \mathcal{B} \setminus \mathcal{B}_c$, the board frame. To model this variant of the problem, constraint set (2.5) of $\mathrm{P_{NL}}$ is modified as:

$$\sum_{(kpq)\in I_{ij}} x^k_{pq} \;=\; 1,\ \forall\ (i,j) \in \mathcal{B}_c \qquad \text{and} \qquad \sum_{(kpq)\in I_{ij}} x^k_{pq} \;\leq\; 1,\ \forall\ (i,j) \in \mathcal{B}_f.$$

2.2.2 Linearized Model

In this section we derive a linearization of P_{NL} using value disjunction reformulation. Observe that values of r_i and c_j are bounded above by $T = \frac{mn}{\ell}$. Thus, each r_i and c_j can be replaced with $T+1$ new binary variables $\{r_{i0}, r_{i1}, \ldots, r_{iT}\}$ and $\{c_{j0}, c_{j1}, \ldots, c_{jT}\}$, respectively. Variable r_{it} (c_{jt}) takes value 1 iff there are exactly t centers of gravity on row i (column j). Finally, for each row i (column j), one needs to ensure that exactly one of the variables r_{it} (c_{jt}) is set to be 1. The above discussion leads to the following linear binary program denoted as P_L.

$$
\begin{aligned}
\mathrm{P_L}: \quad \max \quad & -\sum_{i=1}^{m}\sum_{t=1}^{T}\left(\frac{t}{2T}\lg\frac{t}{2T}\right) r_{it} - \sum_{j=1}^{n}\sum_{t=1}^{T}\left(\frac{t}{2T}\lg\frac{t}{2T}\right) c_{jt} & & \\
\text{s.to} \quad & \sum_{(kpq)\in I_{ij}} x_{pq}^{k} \;=\; 1 & \forall\ (i,j)\in\mathcal{B} \quad & (2.9)\\
& \sum_{t=1}^{T} t\ r_{it} = \sum_{(kpq)\in R_i} x_{pq}^{k} & \forall\ i \quad & (2.10)\\
& \sum_{t=1}^{T} t\ c_{jt} = \sum_{(kpq)\in C_j} x_{pq}^{k} & \forall\ j \quad & (2.11)\\
& \sum_{t=0}^{T} r_{it} = 1 & \forall\ i \quad & (2.12)\\
& \sum_{t=0}^{T} c_{jt} = 1 & \forall\ j \quad & (2.13)\\
& x_{pq}^{k},\ r_{it},\ c_{jt} \in \{0,1\} & \forall\ i,j,p,q,k,t &
\end{aligned}
$$

The disjunctive constraints (2.10)-(2.11) link r_{it} and c_{jt} to the corresponding value of t. Constraints (2.12)-(2.13) ensure that a single t value is chosen for each i and j.

Proposition 1. *The binary variables r_{it} and c_{jt} in P_L can be relaxed to be nonnegative continuous without losing integrality.*

Proof. Function $\phi(x) = x\lg(x)$ is strictly convex in $(0,1]$ as:

$$\phi''(x) = \frac{1}{x} > 0\ .$$

Next, consider strictly convex function $\phi_T(x) = \frac{x}{T}\lg(\frac{x}{T})$ defined on $x \in (0,T]$. Using constraints of the model ($\sum_{t=1}^{T} t\ r_t = \Delta \le T$ and $\sum_{t=0}^{T} r_t = 1$) and Jensen's Inequality [69], whenever more than one r_t is nonzero we have:

$$
\begin{aligned}
\frac{\Delta}{T}\lg(\frac{\Delta}{T}) \;=\; \phi_T(\Delta) \;&=\; \phi_T\left(\sum_{t=1}^{T} t\ r_t\right)\\
&<\; \sum_{t=1}^{T}\phi_T(t)\, r_t\\
&=\; \sum_{t=1}^{T}\frac{t}{T}\lg\left(\frac{t}{T}\right) r_t.
\end{aligned}
$$

Thus, the lower bound on the left hand side is attained only when $r_\Delta = 1$. Clearly, the right-hand side of (2.10)-(2.11) is integral for any feasible solution of $\mathrm{P_L}$. □

In the remainder of this chapter, we assume that variables r and c in $\mathrm{P_L}$ are relaxed to be nonnegative continuous. For additional discussion on value disjunction approach we refer the reader to [70] and [82].

Note that the formulation can be further improved by tightening the upper bound on r and c using information about $\mathcal{B}$ and $\mathcal{P}$. For instance, one immediate improvement is $r_i \leq \min\{m\,n/\ell,\, n\}$ and $c_j \leq \min\{m\,n/\ell,\, m\}$. If only square tetromino are used, then, $r_i \leq \min\{m\,n/\ell,\, n/2\}$ and $c_j \leq \min\{m\,n/\ell,\, m/2\}$. Another point of view is to restrict r_i and c_j based on optimality conditions. Since an ideal distribution would have $T/(m+n)$ centers of gravity on each row and column so that the resulting probability distribution is uniform, one can constrain values of r and c not to deviate from $T/(m+n)$ for too much. However, as the main concern are the binary variables x, we have not spent too much effort in this direction in our implementation.

2.3 COLUMN GENERATION APPROACH

As $|\mathcal{F}_\ell|$ is exponential in ℓ, the number of binary variables in $\mathrm{P_L}$ also grows exponentially. Such large-scale formulations are challenging for commercial MIP solvers, and branch-and-price based methods are often employed [10, 19, 58, 111, 112]. We discuss a new branching strategy for the exact set covering problem in Section 2.3.1 followed by a column generation approach in Section 2.3.2. In Section 2.3.3, we propose a new duality-based lower-bounding scheme.

2.3.1 A New Branching Strategy

Consider a tiling problem $(\mathcal{B}, \mathcal{P})$, $\mathcal{P} \subseteq \mathcal{F}_\ell$. There are $O(mn|\mathcal{P}|)$ subsets of the ground set $\mathcal{B}$, however, only $T = \frac{mn}{\ell}$ of them is used in any feasible tiling. Hence, standard branching on variables typically creates very long branches when excluding sets , i.e., $x^k_{pq} = 0$, whereas

it ends up with very short branches when including them, i.e., $x^k_{pq} = 1$, which is due to frequently arising infeasibility as an inherent property of the exact set covering problem. The depth of this unbalanced search tree is another concern. In order to mitigate these issues, we propose a new branching strategy that results in a more balanced search tree with a considerably shorter depth. A detailed treatment of branching strategies can be found in [111, 112].

Define $\mathcal{M}^k_{pq} = \{s \mapsto (i,j) : 1 \leq s \leq \ell;\ s \in \mathbb{Z};\ (i,j) \in f^k_{pq};\ f^k \in \mathcal{P}\}$ to be a bijection from the set of integers $\{1, \ldots, \ell\}$ to the set of squares f^k_{pq} and $(\mathcal{M}^k_{pq})^{-1}$ to be its inverse. Let $I_{ij}[s] = \{(k,p,q) : (i,j) = \mathcal{M}^k_{pq}[s]\}$ be the set of all triplets (i.e., the corresponding polyomino translations f^k_{pq}) that cover location $(i,j) \in \mathcal{B}$ with their square mapped to s. Thus, $I_{ij} = \bigcup_s I_{ij}[s]$. Next, assume at some node of the branch-and-bound (B&B) tree we have

$$0 < \sum_{(kpq) \in I_{ij}[s]} x^k_{pq} < 1 \tag{2.14}$$

for some (i,j) and s. Then our branching strategy proceeds as follows:

$$x^k_{pq} = 0 \qquad \forall (k,p,q) \in I_{ij}[s], \qquad \text{(Left Branch)}$$

$$x^k_{pq} = 0 \qquad \forall (k,p,q) \in I_{ij} \setminus I_{ij}[s]. \qquad \text{(Right Branch)}$$

All the sets having (i,j) mapped to s are removed from consideration on the left branch, whereas only these set are allowed on the right branch. On both branches we fix some sets of variables to zero. This shortens the depth of the tree compared to fixing only one variable to 0. Also, both branches have some sets of variables that can take positive values, thus, avoiding premature fathoming of the branch when compared to fixing a single variable to 1. Consequently, the proposed branching strategy significantly balances the search tree.

The fact that each $(i,j) \in \mathcal{B}$ must be covered by exactly one polyomino implies that every integral solution x of (2.9) must violate (2.14) for every (i,j) and s. The question is whether it is sufficient, i.e., does every fractional solution satisfies (2.14) for some (i,j) and s?

Proposition 2. *Consider a fractional solution x to (2.9) and let $x_1 > 0$ and $x_2 > 0$ be two nonzero variables associated with polyominoes f^1 and f^2, respectively, that cover $(i,j) \in \mathcal{B}$. Then there exists two bijections $\mathcal{M}^1$ and $\mathcal{M}^2$, such that (2.14) is satisfied for (i,j) for some s.*

Proof. Observe that there are no fractional solutions for monominoes as $|I_{ij}| = 1$ for all (i, j), thus assume $|f| > 1$. Choose two bijections $\mathcal{M}^1$ and $\mathcal{M}^2$ for polyominoes f^1 and f^2 such that $(\mathcal{M}^1)^{-1}[(i, j)] \neq (\mathcal{M}^2)^{-1}[(i, j)]$. Then (2.14) is satisfied for (i, j) when $s = (\mathcal{M}^1)^{-1}[(i, j)]$ because $0 < x_2 < 1$ and f^2 covers (i, j) with $(\mathcal{M}^2)^{-1}[(i, j)]$. □

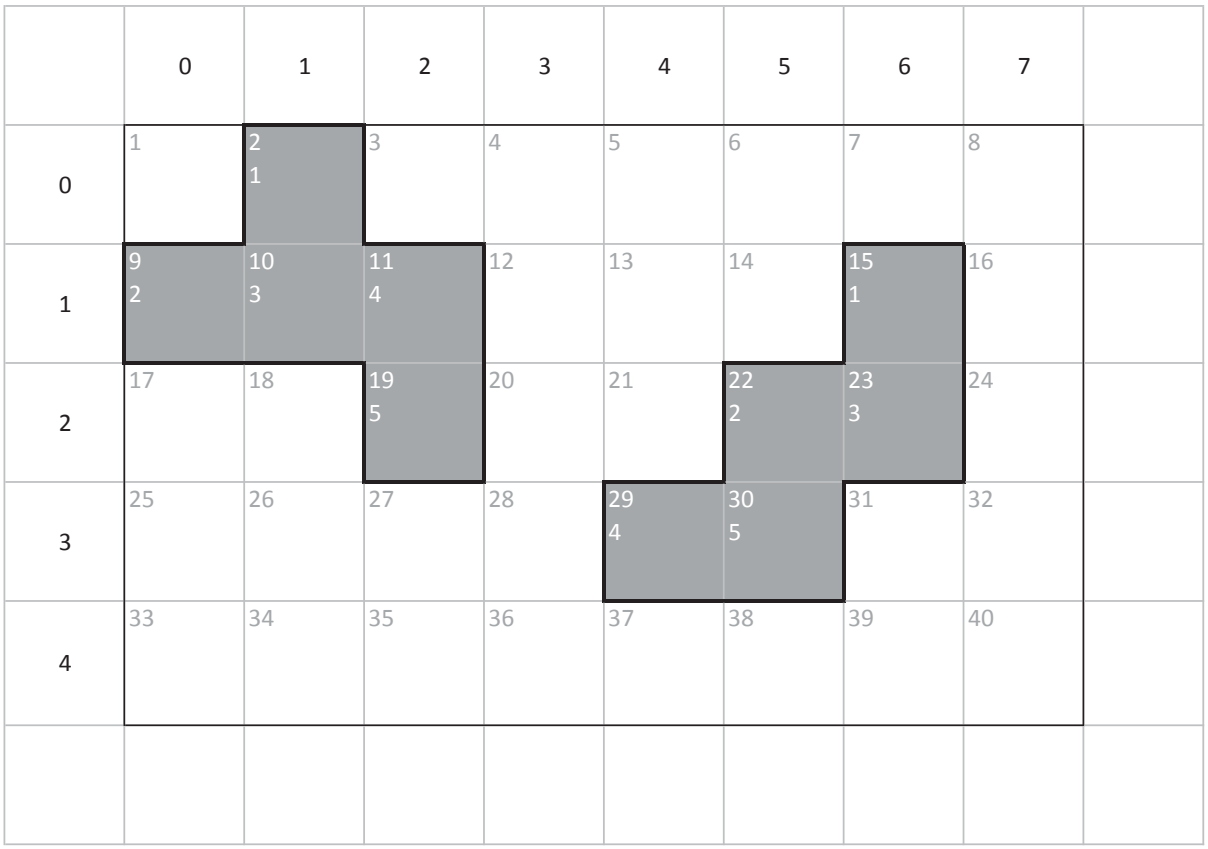
In the worst case, an exponential number of bijections may need to be considered to ensure integrality. However, in our implementation we follow a practically more convenient approach and fix a single bijection to be used for all polyominoes. First, let $\mathcal{M}^{\mathcal{B}} = \{s \mapsto (i, j) \ : \ s = i\,n + j + 1; (i, j) \in \mathcal{B}\}$ be a bijection from the set of integers $\{1, 2, \ldots, m\,n\}$ to $\mathcal{B}$, i.e., the elements of the board are numbered consecutively as seen in Figure 6a. Let $\langle f^k_{pq} \rangle$ be a tuple with elements of the set f^k_{pq} ordered in increasing value of $(\mathcal{M}^{\mathcal{B}})^{-1}$ and denote by $\langle f^k_{pq} \rangle[s]$ the s^{th} element of the tuple. Then

$$(\mathcal{M}^{\mathcal{B}})^{-1}\left[\langle f^k_{pq} \rangle[1]\right] \leq (\mathcal{M}^{\mathcal{B}})^{-1}\left[\langle f^k_{pq} \rangle[2]\right] \leq \ldots \leq (\mathcal{M}^{\mathcal{B}})^{-1}\left[\langle f^k_{pq} \rangle[\ell]\right].$$

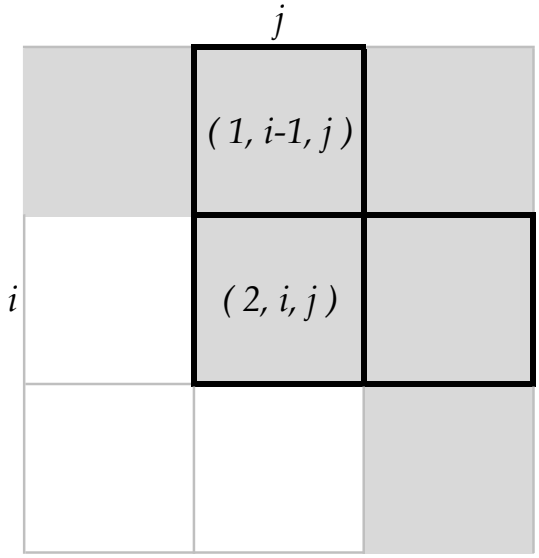
Define $(\mathcal{M}^k_{pq})^{-1}$ to map each element of the set f^k_{pq} to its position in the tuple $\langle f^k_{pq} \rangle$. Hence, $\mathcal{M}^k_{pq}[s] = \langle f^k_{pq} \rangle[s]$ for all (k, p, q). The resulting bijections for F- and W-pentominoes are shown in Figure 6a. For our implementation, we employ this simple bijection. Theoretically, we may branch on all (i, j) and s, and still have a fractional solution. However, as observed through our computational experiments, it is not a common occurrence, especially for larger polyominoes with complex boundary shapes rather than simple dominoes.

Proposition 3. *Branching on $s = 1$ only is sufficient to guarantee integrality for dominoes.*

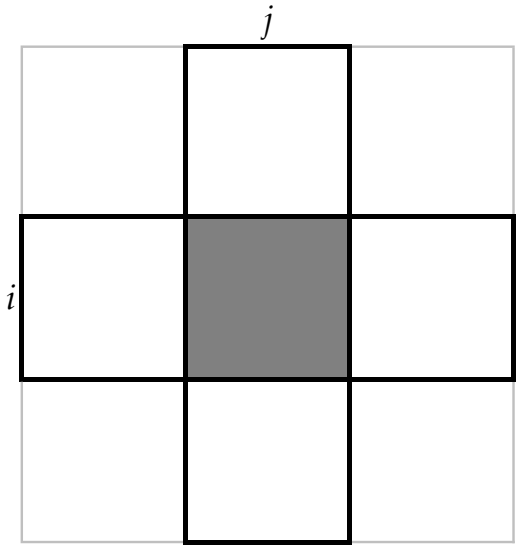
Proof. There is only one domino, $\mathcal{F}_2$, as shown in Figure 1; let its vertical orientation be f^1 and the horizontal orientation be f^2. Assume on the contrary that for a board $\mathcal{B}$, there are fractional x variables but further branching is not possible since no $(i, j) \in \mathcal{B}$ satisfies (2.14) for $s = 1$. Choose a location (i, j) covered by at least two dominoes such that $(i, j - 1)$ and $(i + 1, j)$ are either covered by exactly one domino or they do not belong to $\mathcal{B}$ (i.e., $j - 1 < 0$ or $i + 1 \geq m$). For a finite board, such (i, j) can always be found by selecting a location covered by at least two dominoes and then moving left or down until required condition is met, see Figure 6b. Then, we must have $x^1_{i-1,j} > 0$, $x^2_{i,j} > 0$, and $x^1_{i-1,j} + x^2_{i,j} = 1$.



(a) Bijections for board, F-pentomino, and W-pentomino.



(b) Location (i, j) covered by $f^1_{i-1,j}$ and $f^2_{i,j}$.



(c) Location (i, j) covered by $f^1_{i-1,j}$, $f^1_{i,j}$, $f^2_{i,j-1}$, and $f^2_{i,j}$.

Figure 6: The bijections we use and example fractional domino tilings.

As $\langle f^1_{i-1,j}\rangle[2] = \langle f^2_{i,j}\rangle[1] = (i,j)$, then (2.14) is satisfied for $s = 1$ and (i,j), which is a contradiction. □

The depth of the B&B tree for $\mathrm{P_L}$ in traditional branching is $O(m\,n\,|\mathcal{P}|)$, $\mathcal{P} \subseteq \mathcal{F}_\ell$. By branching on (s,i,j) instead, the depth of the tree becomes $O(m\,n\,\ell)$, which is a substantial improvement if one recalls that $\mathcal{F}_\ell$ is exponential in ℓ. In all our experiments on using our branching strategy, it has been sufficient for most of the time to prune a branch after branching on $s = 1$ only. Fixing $\frac{mn}{\ell}$ locations that should be covered by the first square of the polyominoes along a branch of the B&B tree, with respect to previously discussed bijections, is often sufficient to fathom the branch.

In our implementation, starting from location $(0,0)$ of the board, we branch on the first location (i,j) with an s satisfying (2.14). If there are multiple s values on which to branch, then the one with maximum $\sum_{(kpq)\in I_{ij}[s]} x^k_{pq}$ is selected, so that one of the resulting branches is minimally infeasible. Based on our experiments, this strategy dominates other options as it is able to result in a feasible tiling faster.

Although the above results are in the context of the polyomino tiling problem, they are directly applicable to the general set partitioning setting. Thus, it is necessary to compare the proposed approach to the well-known branching strategy of [103], which relies on the following observation. For every fractional solution x to the set partitioning problem, there exists at least one pair of squares (i,j) and (i',j') such that $0 < \sum_{k\in S_{(ij),(i'j')}} x_k < 1$, where $S_{(ij),(i'j')}$ is the index set for subsets covering both (i,j) and (i',j'). Then, $\sum_{k\in S_{(ij),(i'j')}} x_k = 1$ is enforced on one branch and $\sum_{k\in S_{(ij),(i'j')}} x_k = 0$ on the other branch. Applied to the tiling problem with dominoes only, this branching rule reduces to the single-variable branching because dominoes covering the same two squares cannot be distinct, i.e., $|S_{(ij),(i'j')}| \leq 1$ for all pairs (i,j) and (i',j'). However, in a fractional solution, a single square (i,j) can be covered by as many as four distinct dominoes at the same time, see Figure 6c. Thus, $|I_{ij}[s]| = 2$ and multiple variables appear in (2.14).

2.3.2 Delayed Column Generation (DCG)

Define a *dummy* polyomino f^0 to be a copy of $\mathcal{B}$ with no center of gravity. Observe that f^0 does not contribute to the objective function and induces a single set f^0_{00}, which is also a feasible solution to the problem. Let $I = \bigcup_{(i,j)\in\mathcal{B}} I_{ij} = \{(k,p,q) : f^k_{pq} \subseteq \mathcal{B}, f^k \subseteq \mathcal{P}\}$ be the set of all feasible triplets for $(\mathcal{B},\mathcal{P})$, and $I_0 = \{(0,0,0)\}$. As each triplet defines a column of the original problem, we refer to them as columns.

Denote by $\mathrm{P_L}(\mathcal{B},\tilde{I})$ and $\overline{\mathrm{P}}_\mathrm{L}(\mathcal{B},\tilde{I})$ the exact set covering formulation $\mathrm{P_L}$ for $\mathcal{B}$ using only a set of columns $\tilde{I} \subseteq I$ and its linear programming (LP) relaxation, respectively. We refer to $\overline{\mathrm{P}}_\mathrm{L}(\mathcal{B},\tilde{I}\cup I_0)$ as the *master problem* of $\overline{\mathrm{P}}_\mathrm{L}(\mathcal{B},I)$. In the master problem, variable x^0_{00} for the dummy polyomino is treated as an artificial variable and penalized in the objective using the big-M method. In the following description of one iteration of DCG, *pricing* a set of columns simply corresponds to computing their reduced costs for a given dual vector.

1. Let (x,r,c) be an optimal solution to $\overline{\mathrm{P}}_\mathrm{L}(\mathcal{B},\tilde{I}\cup I_0)$. Extend x to $\hat{x} = (x,0)$ by setting $x^k_{pq} = 0$ for $x^k_{pq} \in \hat{x} \setminus x$ so that $(\hat{x},r,c)$ is a solution of $\overline{\mathrm{P}}_\mathrm{L}(\mathcal{B},I)$.
2. Let π be a dual vector for $\overline{\mathrm{P}}_\mathrm{L}(\mathcal{B},\tilde{I}\cup I_0)$. Price the columns $I\setminus\tilde{I}$. If there are no positive reduced cost columns, then $(\hat{x},r,c)$ is an optimal solution of $\overline{\mathrm{P}}_\mathrm{L}(\mathcal{B},I)$, stop.
3. Let $\hat{I} \subseteq I\setminus\tilde{I}$ be a set of positive reduces cost columns. $\tilde{I} \leftarrow \tilde{I}\cup\hat{I}$. Return to *Step 1*.

We refer the reader to [19, 111] for detailed discussion of the standard branch-and-price approach. Specifically, columns are generated at each node until the master problem for the node is optimal. The optimal objective value of the master is an upper bound for the branch and if it is less than the current global lower bound (i.e., value of the incumbent solution), then the branch is fathomed. If the solution is integral, its objective value is a lower bound for the tree and if it is less than the current global lower bound, then the branch is fathomed. Otherwise incumbent is replaced with the new solution, global lower bound is updated, and the branch is fathomed.

If the optimal integral solution uses only the dummy polyomino, then it implies that the branch is infeasible and it is fathomed. On the other hand, if optimal solution is fractional, branching takes place. In fact, any integral solution obtained during each iteration of the DCG procedure is stored if it improves the global lower bound. Moreover, an upper bound for

the branch is calculated at each iteration of the DCG procedure and the branch is fathomed as soon as it is no longer promising. Details of this upper bound are given in the next section.

For efficiency purposes, we keep a list I_t^n of columns that are eligible for pricing in iteration t at node n of the the B&B tree. The list of eligible columns for the first iteration at the root node includes all feasible columns, i.e., $I_1^0 = I$. Reduced costs of x_{pq}^k, $(k,p,q) \in I_t^n$, are computed using a given dual vector π for $\overline{\mathrm{P}}_{\mathrm{L}}(\mathcal{B}, \tilde{I})$, the master problem in iteration $t-1$ at node n. All columns $\hat{I}$ that are added to the master after iteration t are removed from further consideration: $I_{t+1}^n = I_t^n \setminus \hat{I}$.

The branching strategy proposed in Section 2.3.1 is used in our implementation. Let I^n be the set of triples eligible for pricing after the last iteration of the n^{th} node. If the current solution to the master problem is not integral, we branch on (s,i,j) satisfying (2.14) and set all variables in $I_{ij}[s]$ to be zero on the left branch; thus, $I_1^{n+1} \leftarrow I^n \setminus I_{ij}[s]$ for the left child and $I_1^{n+2} \leftarrow I^n \setminus (I_{ij} \setminus I_{ij}[s])$ for the right child. Consequently, each branching removes many columns from consideration even before they are introduced to the master problem, which improves the search significantly.

Setting variables to zero causes loss of feasibility (since some of them are nonzero); after the LP solver restores feasibility at the end of the first iteration of the new node, possibly by re-introducing x_{00}^0 into the basis, then any variable x_{pq}^k fixed to zero by branching is removed from the master problem. Thus, branching also removes many inactive columns from the master problem on the two new branches, which stabilizes the size of the master problem along all the branches of the tree. Since the size of I_{ij} is usually large, the proposed branching strategy is very effective. The dummy column x_{00}^0 is neither removed from the master problem nor branched on at any time. This is necessary as otherwise infeasibility may arise after branching on the original variables $\{x_{pq}^k : k > 0\}$, which would cause problem in getting a dual vector for the master.

Another approach that potentially speeds up the algorithm is to not generate any columns, except at the root node, as long as branching is possible. Specifically, (i) for the root node, price columns until the master problem is optimal, (ii) if either $x_{00}^0 > 0$, or the current solution is integral but not optimal, then price columns in the current iteration, and (iii) in any other scenario, continue branching.

The main reason to solve the root node to optimality is to obtain a true upper bound for the B&B tree. If $x^0_{00} > 0$, then columns have to be generated as otherwise branching would fail. If the current solution is integral, we would like to make sure that it is optimal so that we can either fathom the node or continue branching. In any other case, columns are priced in the first iteration of the node to update the upper bound for the branch; but immediately after the first iteration branching takes place, without adding any columns to the master problem, as long as the obtained upper bound does not lead to fathoming of the branch.

2.3.3 A New Lower-Bounding Scheme

We improve the lower-bounding scheme proposed by [58] within the context of branch-and-price for solving the exact set covering problem. Unlike [58], our lower bound does not depend on the maximum number of sets that form a partition. Since the proposed bound is applicable to the general exact set covering problem, we follow the notation from [58] in this section.

Consider an m-element ground set to be covered (exactly) and n subsets $\{f_1, f_2, \ldots, f_n\}$ of it. Associate a binary variable x_j with the subset f_j and let c_j denote its cost. Then the generic exact set covering problem is formulated as

$$\text{ES} : \min \left\{ c^T x \; : \; \sum_{j \in N} f_j \, x_j = e; \; x \in \{0,1\}^{|N|} \right\},$$

where e is the m-dimensional vector of ones and $N = \{1, 2, \ldots, n\}$ is the index set for all the subsets of the ground set. In the context of column generation, the restricted master problem ES_M is the LP relaxation of ES written for some subset $\hat{N} \subset N$:

$$\text{ES}_\text{M} : \min \left\{ \sum_{j \in \hat{N}} c_j x_j \; : \; \sum_{j \in \hat{N}} f_j \, x_j = e; \; x \geq 0 \right\}.$$

Associating an m-dimensional dual vector $\hat{\pi}$ with the exact covering constraints of ES_M, the corresponding complementary column generation subproblem, ES_S, is given as

$$\text{ES}_\text{S} : \min \left\{ \sum_{j \in N \setminus \hat{N}} (c_j - f_j^T \hat{\pi}) \, x_j \; : \; \sum_{j \in N \setminus \hat{N}} f_j \, x_j \leq e; \; x \in \{0,1\}^{|N \setminus \hat{N}|} \right\}.$$

Similarly, associating an m-dimensional dual vector $\hat{y}$ with the set packing constraints of $\mathrm{ES_S}$, the dual of LP relaxation of $\mathrm{ES_S}$, referred to as $\overline{\mathrm{ES}}_\mathrm{S}$, is

$$\overline{\mathrm{ES}}_\mathrm{S} : \max\left\{e^T\hat{y} \ : \ f_j^T\hat{\pi} + f_j^T\hat{y} \leq c_j, \forall\, j \in N \setminus \hat{N};\, \hat{y} \leq 0\right\}.$$

Proposition 4. *Let $\hat{\pi}$ be any dual vector to the partitioning constraints of the formulation ES_M and $\hat{y}$ be a solution to $\overline{ES}_S$ for $\hat{\pi}$. Then*

$$e^T(\hat{\pi} + \hat{y})$$

is a lower bound for the LP relaxation of ES.

Proof. Consider the following equivalent reformulation of ES:

$$\mathrm{ES}' : \min\left\{c^Tx \ : \ \sum_{j\in N} f_jx_j = e;\, \sum_{j\in N} f_jx_j \leq e;\, x \in \{0,1\}^{|N|}\right\}.$$

Let π and y be dual vectors for partitioning and packing constraints of the LP relaxation of ES', respectively. Then dual of ES' is

$$\overline{\mathrm{ES}}' : \max\left\{e^T(\pi + y) \ : \ f_j^T\pi + f_j^Ty \leq c_j,\, \forall\, j \in N;\, \pi \ u.r.s.,\, y \leq 0\right\}.$$

$\overline{\mathrm{ES}}_\mathrm{S}$ and $\overline{\mathrm{ES}}'$ share the same constraints for $j \in N \setminus \hat{N}$. For $j \in \hat{N}$, we have that $f_j^T\hat{y} \leq 0$ as $\hat{y} \leq 0$ and $c_j - f_j^T\hat{\pi} \geq 0$ as $\hat{\pi}$ is a dual feasible solution of $\mathrm{ES_M}$. Therefore, $(\hat{\pi}, \hat{y})$ is feasible to $\overline{\mathrm{ES}}'$. □

Next we show that the lower bound $e^T(\hat{\pi} + \hat{y})$ is superior to the bound proposed by [58]. Consider the LP relaxation of the following equivalent reformulation of ES:

$$\min\left\{c^Tx \ : \ \sum_{j\in N} f_jx_j = e;\, \sum_{j\in N} x_j \leq k;\, x \in \{0,1\}^{|N|}\right\},$$

where k is assumed to be an implied logical/optimality bound on the number of subsets that can be used by any feasible exact set covering. Let the current minimum reduced cost $\delta = \min\{c_j - f_j^T\hat{\pi} \ : \ j \in N\}$ with respect to the restricted master problem be nonpositive. Similarly, $\hat{\pi}$ is a dual feasible vector for the restricted master. One can also remove the set of indices $\hat{N}$, corresponding to the columns currently in the restricted master, from

consideration when calculating δ as they have nonnegative reduced cost with respect to $\hat{\pi}$. Then $(e^T\hat{\pi} + k\delta)$ is a valid lower bound for the LP relaxation of ES [58]. Note that nonnegativity of δ would imply that $\hat{\pi}$ is in fact feasible to the dual of LP relaxation of ES and thus $e^T\hat{\pi}$ is the optimal objective value of the LP relaxation of ES.

Proposition 5. *Let $\hat{\pi}$ be any dual vector to the partitioning constraints of the formulation ES_M and $\hat{y}$ be a solution to $\overline{ES}_S$ for $\hat{\pi}$. Then*

$$e^T\hat{y} \geq k\delta\,.$$

Proof. Let $\delta = \min\{c_j - f_j^T\hat{\pi} \,:\, j \in N \setminus \hat{N}\}$ be nonpositive. Denote by $\mathrm{ES_S}(\delta)$ and $\overline{\mathrm{ES}}_\mathrm{S}(\delta)$ the formulations $\mathrm{ES_S}$ and $\overline{\mathrm{ES}}_\mathrm{S}$ where each reduced cost $c_j - f_j^T\hat{\pi}$ is replaced by δ. Furthermore, let $\mathrm{ES}_\mathrm{S}^{LP}(\delta)$ be LP relaxation of $\mathrm{ES_S}(\delta)$. Clearly $\overline{\mathrm{ES}}_\mathrm{S}(\delta)$ is the dual of $\mathrm{ES}_\mathrm{S}^{LP}(\delta)$. Then it follows that $opt(\overline{\mathrm{ES}}_\mathrm{S}(\delta)) = opt(\mathrm{ES}_\mathrm{S}^{LP}(\delta)) \geq k\delta$ as $\sum_{j\in N} x_j \leq k$. Since polyhedron for $\overline{\mathrm{ES}}_\mathrm{S}(\delta)$ is included in polyhedron for $\overline{\mathrm{ES}}_\mathrm{S}$, $opt(\overline{\mathrm{ES}}_\mathrm{S}) \geq opt(\overline{\mathrm{ES}}_\mathrm{S}(\delta))$. □

As it is not desirable to solve a mathematical program each time the lower bound is calculated, we propose a heuristic procedure to solve $\overline{\mathrm{ES}}_\mathrm{S}$ in the context of the polyomino tiling problem. It is clear that any feasible solution of $\overline{\mathrm{ES}}_\mathrm{S}$ also leads to a valid lower bound. In our implementation, we use the fact that all the polyominoes used in the tiling have the same size. The algorithm is essentially a greedy heuristic and its details are provided in Algorithm 1.

Proposition 6. *Algorithm 1 returns a valid lower bound.*

Proof. To show its correctness, i.e., that it returns a feasible solution, assume j' is the last subset considered by Algorithm 1. For j', $\hat{y}_i = \frac{r_j}{\ell} \leq \frac{r_{j'}}{\ell}$ for every $i \in I^-$ as value of $\hat{y}_i$ must have been set by a previous subset j and $r_j \leq r_{j'}$. Therefore, $\sum_{i\in\hat{f}_{j'}} \hat{y}_i \leq r_{j'}$. This is true for all $0 \leq j \leq j'$. If there are more sets to consider, then it must be the case that all $\hat{y}_i$ are already considered. Therefore, $\sum_{i\in\hat{f}_j} \hat{y}_i \leq r_j$ for all $j > j'$. □

Algorithm 1: Bounding Heuristic

Input: (i) a $1 \times (m_1 + m_2)$ dual vector $\hat{\pi}$ and (ii) an index set $\overline{N}$ for negative reduced cost variables x_j, their corresponding columns f_j, $|f_j| = m_1 + m_2$, and the set $\hat{f}_j$ of locations covered by subset j, $|\hat{f}_j| = \ell$.

Let J be a tuple of elements of the set $\overline{N}$ sorted in increasing order of the reduced cost $r_j = c_j - f_j^T \hat{\pi} = -f_j^T \hat{\pi}$.

Set $\hat{y}_i = 0$ for all i. Set $j = 0$ and $L = 0$.

while $j < |\overline{N}|$ **and** $L \leq m_1$ **do**

  For $i \in \hat{f}_j$, let I^0 and I^- be the index sets of $\hat{y}_i = 0$ and $\hat{y}_i < 0$, respectively.

  if $|I^0| > 0$ **then**

    Set $\hat{y}_i = \frac{r_j}{\ell}$ for $i \in I^0$.

  $j \leftarrow j + 1$ and $L \leftarrow L + |I^0|$.

return $\sum_i \hat{y}_i$

2.4 HEURISTIC APPROACHES

Large-scale polyomino tiling is a difficult combinatorial optimization problem. This section focuses on heuristic approaches since exact methods described in Sections 2.2 and 2.3 can generate solutions only for small- and medium-size problems. Specifically, in Section 2.4.1, we present two algorithms that construct tilings for large boards using exact solutions of smaller boards. Obtained solutions can be further improved using methods described in Section 2.4.2.

2.4.1 Construction Heuristics

2.4.1.1 Zoom-in Algorithm (ZiA): This algorithm is motivated by the *tiling ability* of polyomino sets as defined by [59, 60]. A polyomino set $\mathcal{P}$ is said to have the *strong rep-tile* (repetitive tiling) property if every member of $\mathcal{P}$ can be tiled using $\mathcal{P}$ at some common scale $a \times b$, i.e., each square of each piece is replaced with an $a \times b$ rectangle to obtain its

larger-scale model. We refer to the $a \times b$ scale as the *zoom level*. Pentomino family has the strong rep-tile property at 5×5 zoom level, see Figure 7. Note that if there exists an exact tiling of an $a \times b$ rectangular board by a polyomino set $\mathcal{P}$, then $\mathcal{P}$ has the strong rep-tile property at $a \times b$ zoom level. However, the statement in the opposite direction is not necessarily true.

Given an initial $m \times n$ board $\mathcal{B}$ and its tiling $\mathcal{T}$ using a set of polyominoes $\mathcal{P} = \{f^1, \ldots, f^K\}$, assume $\mathcal{P}$ has the strong rep-tile property at $a \times b$ zoom level. Then a tiling for an $am \times bn$ board can be obtained by replacing each f^k_{pq} used by $\mathcal{T}$ with $\mathcal{T}^k$, the tiling of f^k at zoom level $a \times b$. Observe that repeating this procedure in an iterative manner, one can obtain extremely large tilings very fast. Specifically, after t iterations, a tiling of the board of size $a^t m \times b^t n$ is readily available. Moreover, tiling of each polyomino at $a \times b$ zoom level has to be constructed only once (using any of the exact approaches) and can be used later on as needed. The initial tiling can be of a relatively small size. The pseudocode for ZiA is given in Algorithm 2. Assuming that initial tilings $\mathcal{T}^1$, ..., $\mathcal{T}^K$ and $\mathcal{T}$ are computed a priori, then the overall complexity of the method is $O(a\,b\,m\,n)$.

Figure 8 shows a 250×250 tiling obtained after two applications of this algorithm on the initial 10×10 tiling of pentominoes using 5×5 zoom level. Outline for the polyominoes in the starting tiling are also shown in the figure. Entropy of the final tiling is only 0.008% away from its theoretical upper bound lg(500).

In Figure 9, we start from a 12×16 exact tiling of *point-up* octomino with 8.41% optimality gap. Then, using the tiling of point-up octomino with point-up octominoes at 12×12 zoom level (the smallest possible zoom level), the original tiling is enlarged to a 144×192 one with 1.34% optimality gap. Next iteration of the procedure results in a 1728×2304 tiling with 0.79% optimality gap. Only a partial drawing of the complete tiling is given due to memory size limitations. Finally, note that ZiA is applicable only if the considered set $\mathcal{P}$ has the strong rep-tile property.

ZiA algorithm as described above is a bottom-up approach. One can further generalize it to obtain a top-down approach. Assume we want to construct an $m \times n$ tiling. Let $m = \Pi_{i=1}^{\omega(m)} m_i^{\alpha_i}$ and $n = \Pi_{i=1}^{\omega(n)} n_i^{\beta_i}$ be the prime factorizations of m and n where $\omega(m)$ ($\omega(n)$)

Algorithm 2: Zoom-in (ZiA)

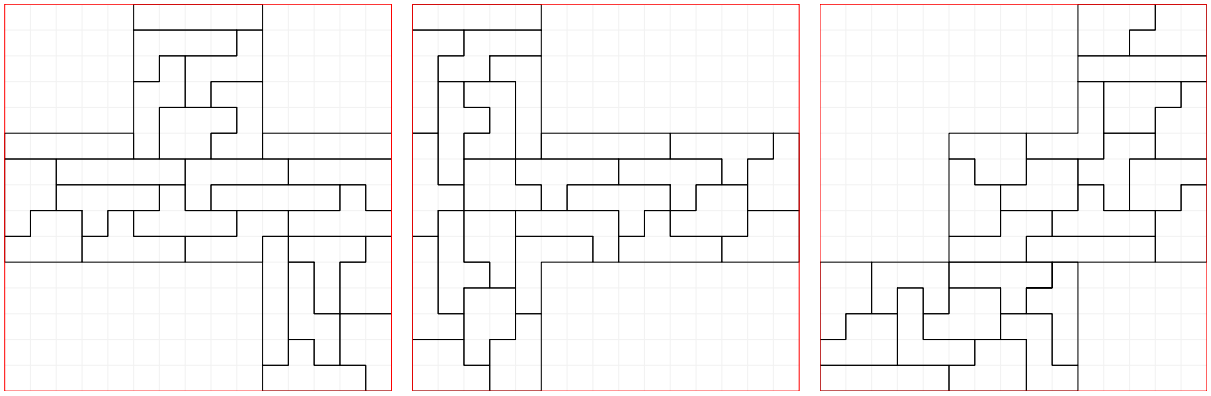
Input: (i) $m \times n$ tiling $\mathcal{T}_0$ using $\mathcal{P}$, (ii) zoom level $a \times b$, and (iii) the exact tilings $\mathcal{T}^k$ for each polyomino $f^k \in \mathcal{P}$ using $\mathcal{P}$ at the zoom level $a \times b$.

$\mathcal{T}_1 \leftarrow \emptyset$

foreach $(k, p, q) \in \mathcal{T}_0$ **do**

  $\mathcal{T}_1 \leftarrow \mathcal{T}_1 \cup \{(\bar{k}, a\,p + \bar{p}, b\,q + \bar{q}) : (\bar{k}, \bar{p}, \bar{q}) \in \mathcal{T}^k\}$ `// place a copy of` $\mathcal{T}^k$ `at` $(a\,p, b\,q)$

return $\mathcal{T}_1$



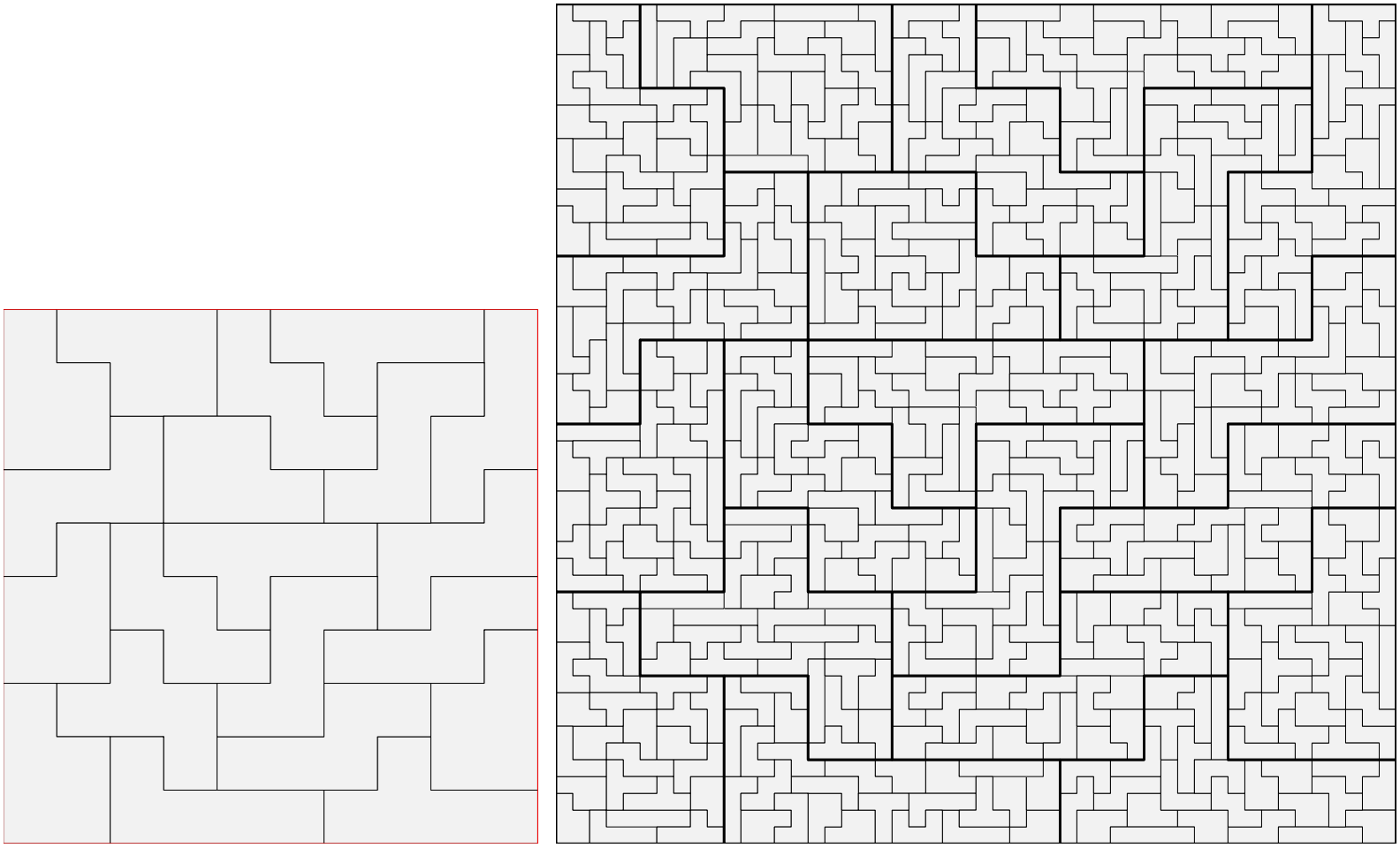
(a) F-pentomino rotated 90°. (b) T-pentomino rotated 270°. (c) W-pentomino.

Figure 7: F-, T-, and W-pentominoes at 5×5 zoom level.

and m_i (n_i) denote the number of distinct prime factors and the distinct prime factors of m (n), respectively. Consider a grouping of $\{m_i\}_{i=1}^{\omega(m)}$ and $\{n_i\}_{i=1}^{\omega(n)}$ into $J+1$ groups

$$\left\{\left(\Pi_{i\in P_j} m_i^{\alpha_i^j}, \Pi_{i\in Q_j} n_i^{\beta_i^j}\right)\right\}_{j=0}^{j=J}$$

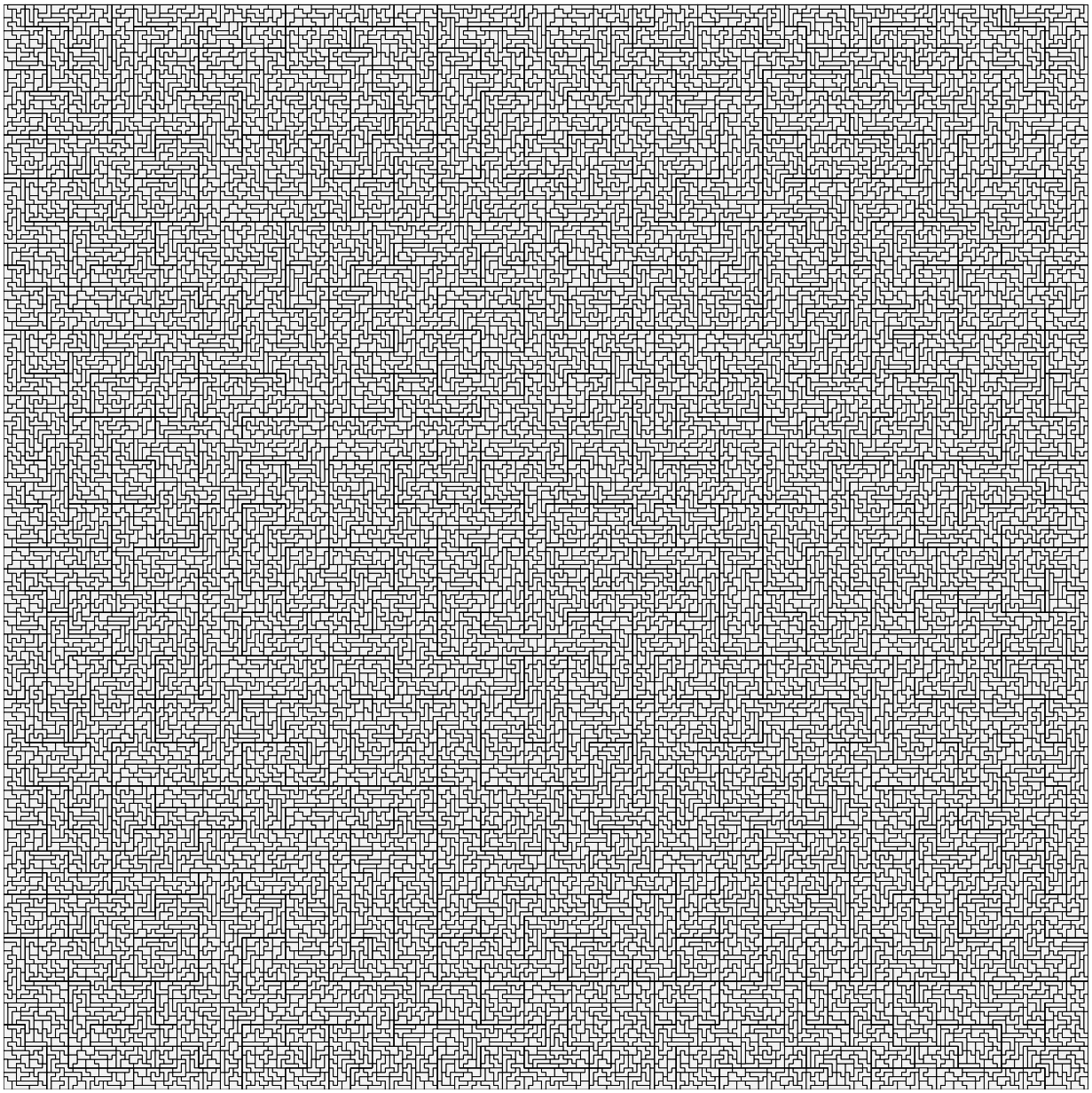
such that $\sum_{j=0}^{J} \alpha_i^j = \alpha_i$ for all m_i, $\sum_{j=0}^{J} \beta_i^j = \beta_i$ for all n_i, and $P_j = \emptyset$ ($Q_j = \emptyset$) implies $\Pi_{i\in P_j} m_i^{\alpha_i^j} = 1$ ($\Pi_{i\in Q_j} n_i^{\beta_i^j} = 1$). If there exists a polyomino set $\mathcal{P}$ that can tile a $\Pi_{i\in P_0} m_i^{\alpha_i^0} \times \Pi_{i\in Q_0} n_i^{\beta_i^0}$, $m_0 \times n_0$, rectangular region and has the strongly rep-tile property for all $\{(\Pi_{i\in P_j} m_i^{\alpha_i^j}, \Pi_{i\in Q_j} n_i^{\beta_i^j})\}_{j=1}^{j=J}$, $\{(a_j, b_j)\}_{j=1}^{J}$, then one can zoom in the $m_0 \times n_0$ tiling for each



(a) 10×10 tiling using pentominoes.

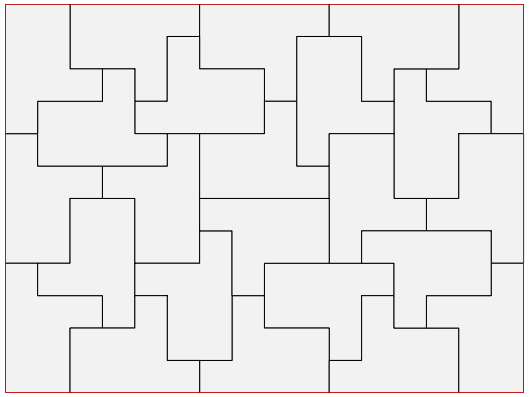
(b) Figure in (a) zoomed-in by 5×5.

Figure 8: Obtaining a 250×250 tiling of pentominoes using after two successive applications of zoom-in heuristic. Solution time is less than one second.

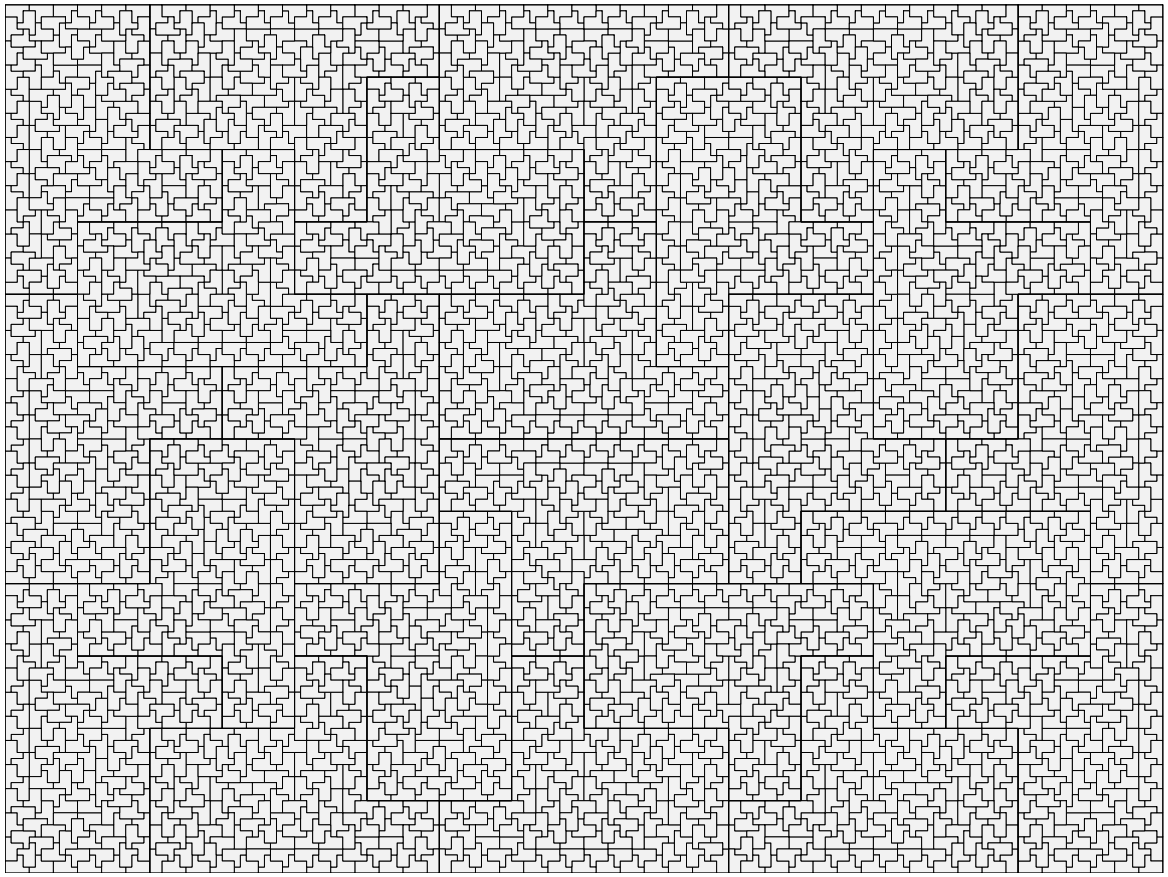


(c) Figure in (b) zoomed-in by 5×5. Gap = 0.008%.

Figure 8: Obtaining a 250×250 tiling of pentominoes using after two successive applications of zoom-in heuristic. Solution time is less than one second.

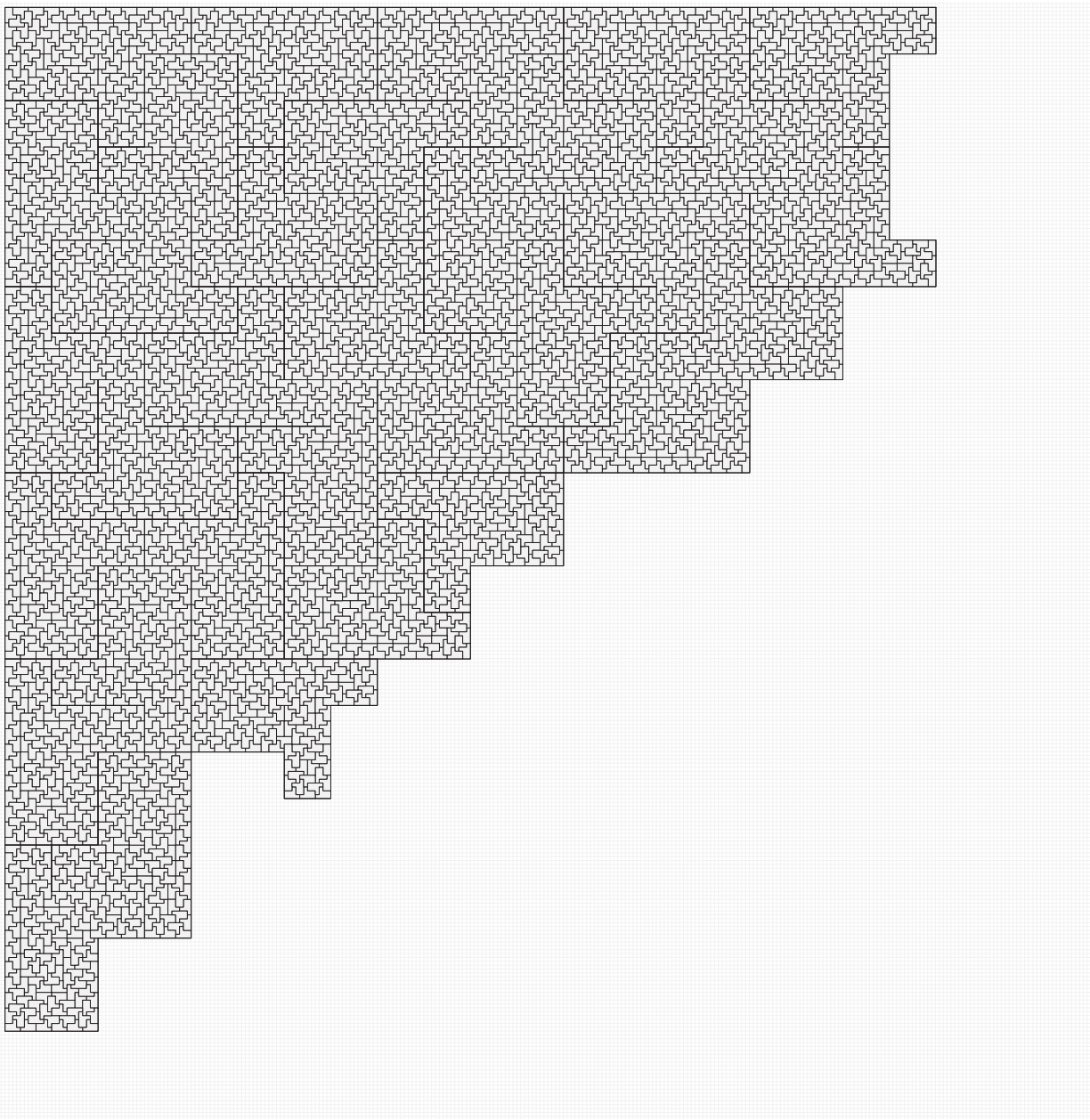


(a) 12×16 L-octomino tiling.



(b) Figure in (a) zoomed-in by 12×12 to obtain 144×192 tiling.

Figure 9: Zooming-in the point-up octomino. The initial 12×16 tiling zoomed-in twice at 12×12 zoom level to obtain a 1728×2304 tiling with 0.69% optimality gap.



(c) A partial drawing of tiling in (b) zoomed-in by 12×12 to obtain a 1728×2304 tiling. Gap = 0.69%

Figure 9: (continued).

$a_j \times b_j$ to obtain an $m \times n$ tiling. To give an example, a 100×100 tiling of pentominoes can be obtained by starting from an initial tiling of a 5×5 board and zooming in for each zoom level $\{(4,4),(5,5)\}$ consecutively.

2.4.1.2 Meta-rectangle Tiling Algorithm (MrTA): This heuristic is motivated by the *meta-rectangle* idea of [114] developed in the context of the classical cutting stock problem. Given a set of polyominoes $\mathcal{P} = \{f^1, \ldots, f^K\}$, assume that there exists a set of rectangle sizes $\mathcal{R} = \{r_1 = (x_1, y_1), \ldots, r_k = (x_k, y_k)\}$ such that every $r \in \mathcal{R}$ can be tiled exactly using polyominoes from $\mathcal{P}$. Similar to the case for ZiA, these rectangular tilings can be obtained a priori using any of the exact approaches. Two rectangles or meta-rectangles (fixed or rotated) are juxtaposed horizontally or vertically to form a new meta-rectangle. Then the obtained meta-rectangle has a polyomino tiling since it is known how to tile each rectangle that constitutes the meta-rectangle.

As demonstrated in Figure 10 and described in Algorithm 3, we propose to construct such meta-rectangle tilings by juxtaposing two rectangles horizontally and then repeating each of them vertically until we reach the least common multiplier (lcm) of the vertical dimensions of the two rectangles. This is the key difference compared to the approach of [114]; it is necessary since we cannot tolerate *waste* (i.e., empty areas that are allowed in the context of the cutting stock problem) in our tilings. Given an initial set of rectangles $\mathcal{R}$ and maximum allowable meta-rectangle dimensions $(\bar{x}, \bar{y})$, MrTA tries to construct all the distinct-size $x \times y$ meta-rectangles ($x \leq \bar{x}$ and $y \leq \bar{y}$) that can be obtained using different rotations of the given rectangles from $\mathcal{R}$. After obtaining the set of all meta-rectangles, one can check if there exists a meta-rectangle with dimensions that are the same as or close to the desired tiling dimensions.

Proposition 7. *Given that $\bar{x} \geq \bar{y}$, computational complexity of Algorithm 3 is $O(\bar{x}^4 \ln^3 \bar{x})$.*

Proof. Algorithm 3 compares two rectangles at most once as only the newly created meta-rectangles are compared to the existing meta-rectangles and as each rectangle is created at most once. There are at most $\bar{x}^2$ possible dimensions for each rectangle and at most one rectangle is created for each possible dimension. Therefore, a maximum of $\bar{x}^4$ comparisons

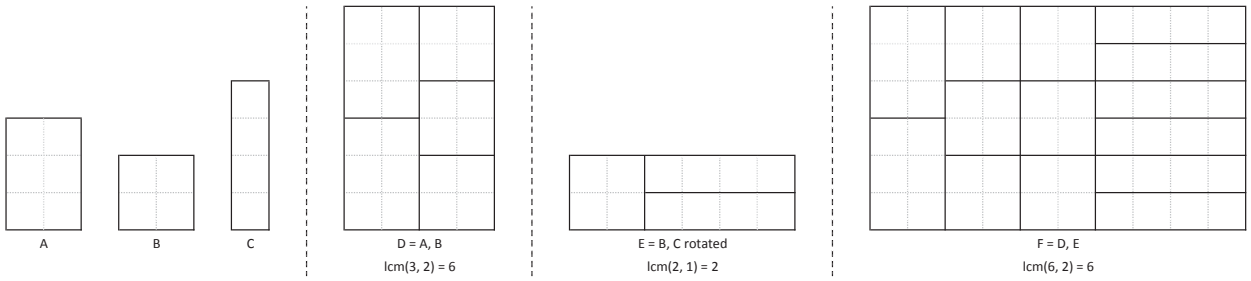


Figure 10: Creating meta-rectangles.

can be performed in total. Checking whether a rectangle exists is constant time when $\mathcal{R}_{all}$, the set of all possible rectangles, is implemented as a two-dimensional array. Since $lcm(a, b) = a\, b/gcd(a, b)$, where gcd stands for greatest common divisor, lcm and gcd have the same computational complexity. Given that $a \geq b$, the complexity of Euclidean Algorithm for $gcd(a,b)$ is $O(\ln^3 a)$ [92]. The result follows. □

Figure 11 shows a 120×160 meta-rectangle and its corresponding octomino tiling, which has 6.33% optimality gap. Figure 12 is a graphical illustration of the output generated by MrTA. Subfigures have horizontal and vertical scales ranging from 1 to 160, where each (x, y) indicates whether there exists a meta-rectangle of dimensions (x, y). Since rectangles (x, y) and (y, x) are the same (rotations), the charts are upper triangular. Each chart shows all the meta-rectangles created starting from different subsets of rectangles. For instance, in Figure 12c, MrTA is initialized with the rectangles $\{(3, 8), (4, 10)\}$, which are known to be tileable with octominoes. It can be observed that even using only two rectangles, MrTA is able to find many distinct size tilings, denoted with black boxes in the figure. Note that if one can provide as many rectangles of different prime number dimensions as possible, MrTA may cover many of the rectangle dimensions on the entire plane, which makes it a highly promising solution heuristic given the complexity of the problem we consider.

One generalization of this algorithm is to consider more than two rectangles at a time. Another improvement is to compare the entropy of newly constructed meta-rectangles with existing ones and keep the ones with the better objective function values. However, it is

Algorithm 3: Meta-rectangle Tiling (MrTA)

Input: (i) maximum allowed meta-rectangle dimensions $(\bar{x}, \bar{y})$, (ii) a set of rectangles $\mathcal{R}$, with $r \in \mathcal{R}$ having dimensions $r = (x, y) \leq (\bar{x}, \bar{y})$, and (iii) a function `lcm(a,b)` which returns the least common multiple of two positive integers (a, b).

$\mathcal{R}_{all} \leftarrow \mathcal{R}$, $\mathcal{R}_{new} \leftarrow \emptyset$

while $\mathcal{R} \neq \emptyset$ **do**

  foreach $r_a \in \mathcal{R}$ *and* $r_b \in \mathcal{R}_{all}$ **do**

    $r_1 = (x_a + x_b,\ \texttt{lcm}(\ y_a, y_b))$ `// ` r_a `and` r_b `fixed`

    $r_2 = (x_a + y_b,\ \texttt{lcm}(\ y_a, x_b))$ `// ` r_a `fixed,` r_b `rotated`

    $r_3 = (y_a + x_b,\ \texttt{lcm}(\ x_a, y_b))$ `// ` r_a `rotated,` r_b `fixed`

    $r_4 = (y_a + y_b,\ \texttt{lcm}(\ x_a, x_b))$ `// ` r_a `and` r_b `rotated`

    foreach $i \in \{1, 2, 3, 4\}$ **do**

      if $r_i \leq (\bar{x}, \bar{y})$ **and** $r_i \notin \mathcal{R}_{all}$ **then**

        $\mathcal{R}_{new} \leftarrow \mathcal{R}_{new} \cup r_i$

  $\mathcal{R} \leftarrow \mathcal{R}_{new}$; $\mathcal{R}_{all} \leftarrow \mathcal{R}_{all} \cup \mathcal{R}_{new}$; $\mathcal{R}_{new} \leftarrow \emptyset$

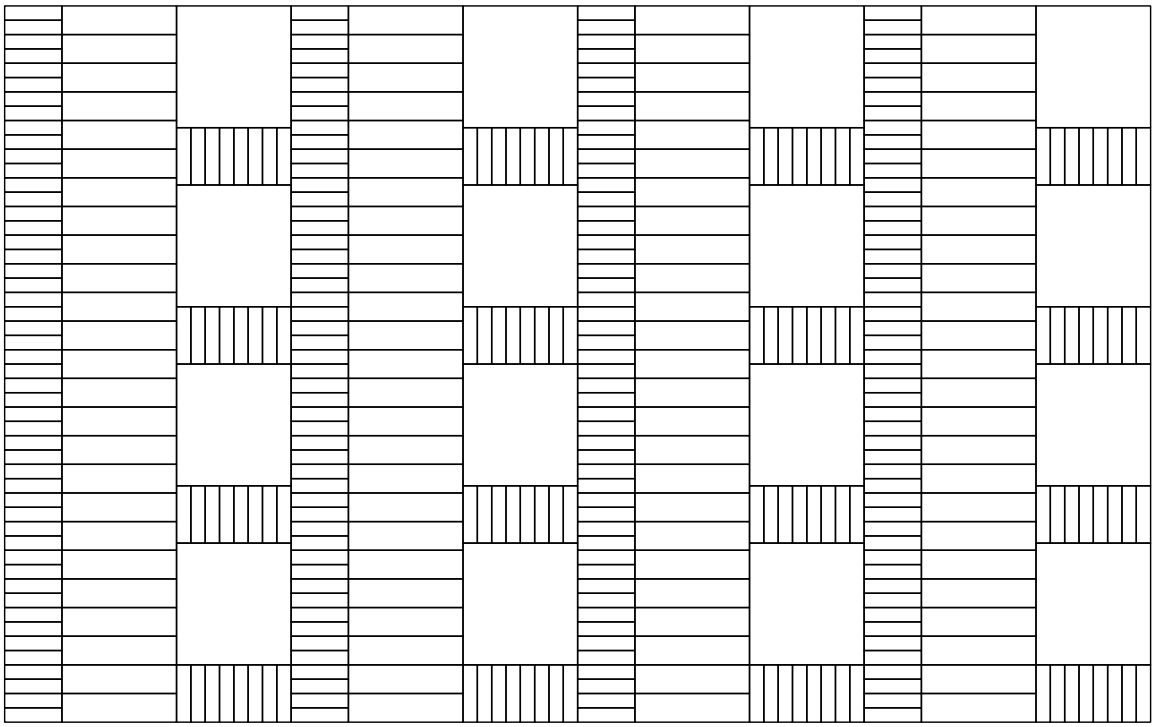
return $\mathcal{R}_{all}$.

evident that these improvements would require recursive update of meta-rectangles, and thus, complexity of the algorithm would increase considerably.

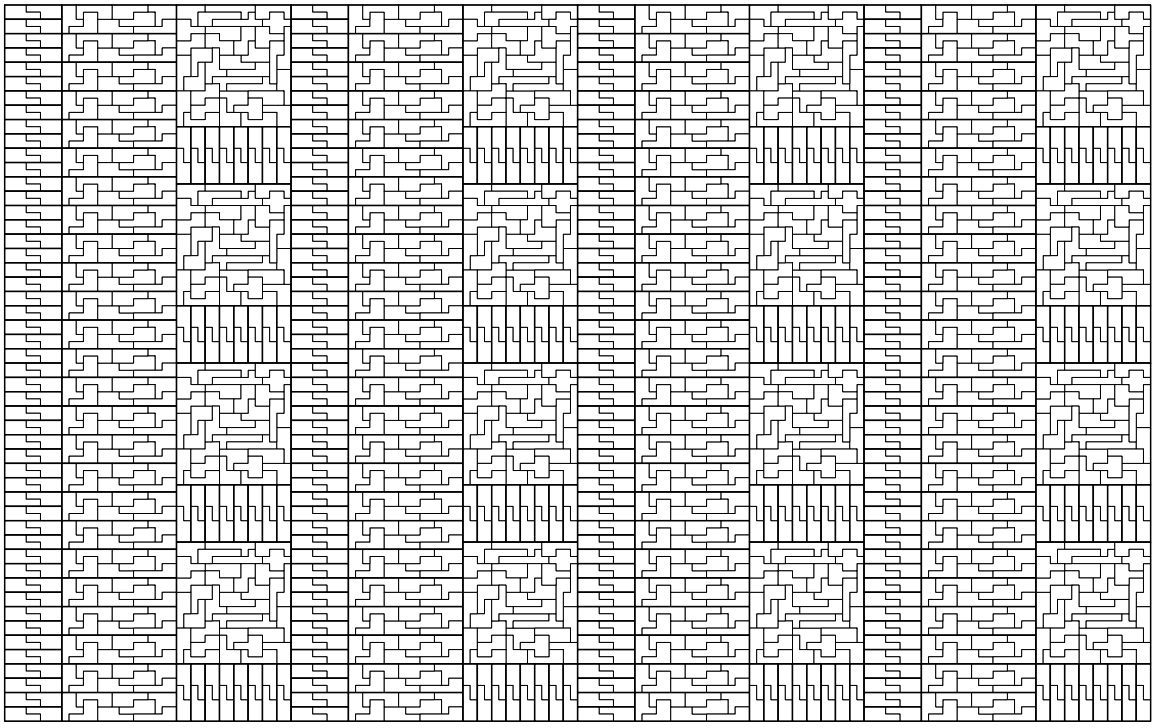
2.4.2 Improvement Heuristics

Next we briefly describe several heuristic procedures designed to improve tilings obtained either by Zoom-in or Meta-rectangle Tiling Algorithms.

Retile: For $m \times n$ board $\mathcal{B}$ and its tiling $\mathcal{T}$ (not necessarily perfect), this procedure retiles a given square sub-region of $\mathcal{B}$ referred to as *window* w of size $(2d+1) \times (2d+1)$ with its center located at the square $(r, c) \in \mathcal{B}$. An example of the initialization step of the Procedure Retile on the corner of an imperfect tiling is shown in Figure 13. Specifically, the procedure (i) retiles

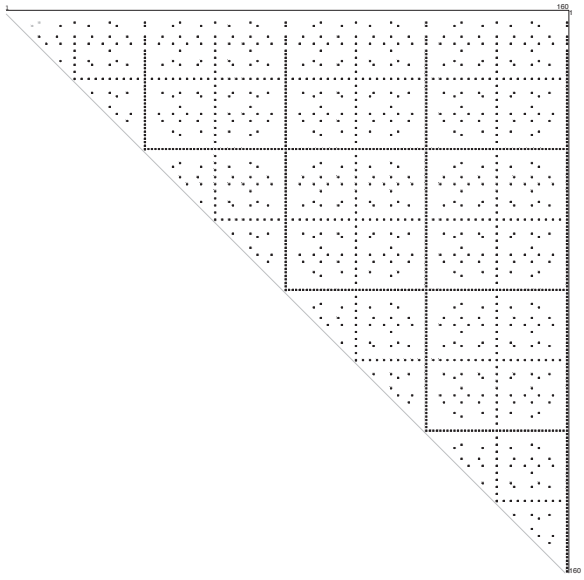


(a) A 100×160 meta-rectangle.

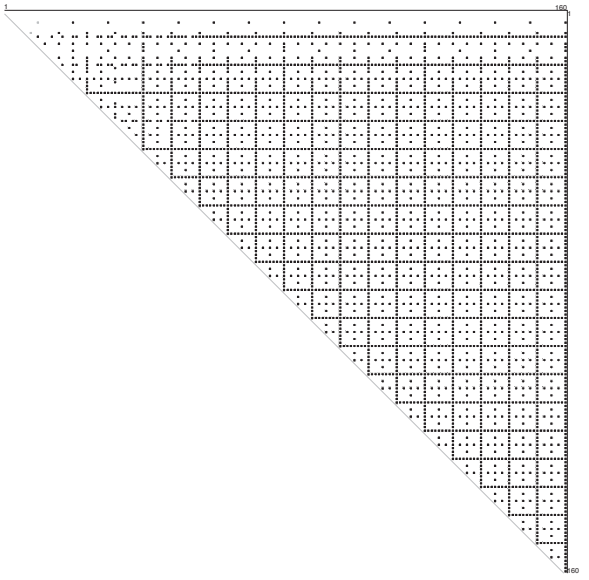


(b) Octomino tiling of the meta-rectangle in (a) with a corresponding 6.33 % optimality gap.

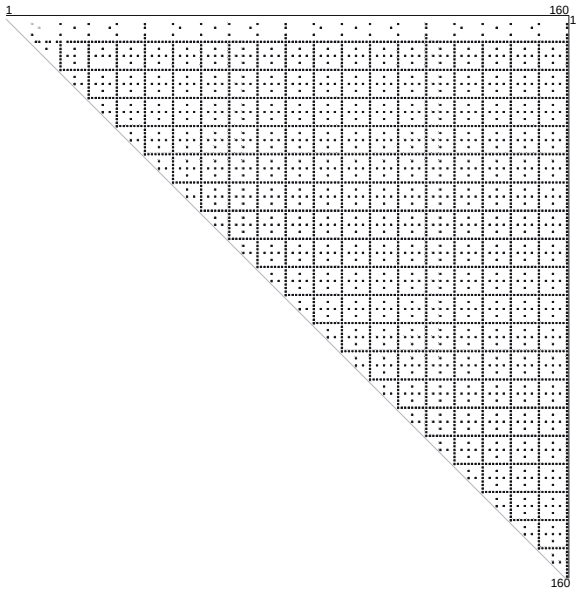
Figure 11: An example of a meta-rectangle generated by Algorithm 3 using initial set of rectangles $\{(2, 8), (4, 16), (16, 17)\}$ and its final octomino tiling.



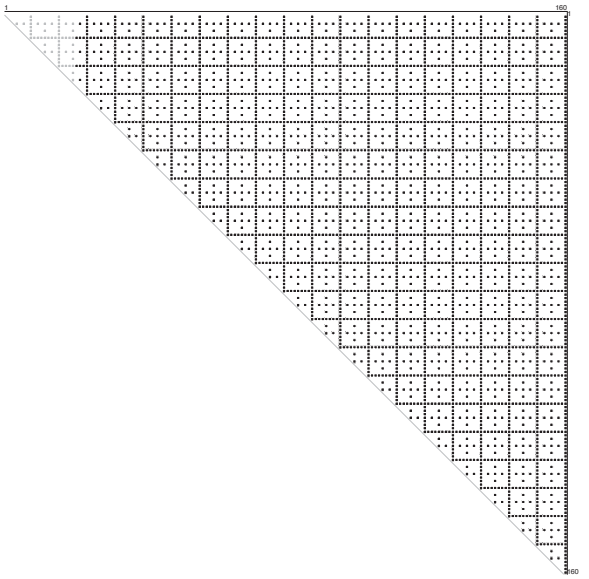
(a) All meta-rectangles created using only $\{(4, 10), (5, 8)\}$.



(b) All meta-rectangles created using only $\{(4, 10), (7, 8)\}$.



(c) All meta-rectangles created using only $\{(3, 8), (4, 10)\}$.



(d) All meta-rectangles created using 55 rectangles tileable with octominoes ($r \leq (20, 20)$).

Figure 12: Dimensions of meta-rectangles generated by MrTA and the initialization step of the tiling algorithm.

squares labeled ‘B’ in order to increase local irregularity, and (ii) penalizes squares labeled ‘C’ in order to obtain a perfect tiling from an imperfect one. Squares labeled ‘A’ should never be covered because polyominoes covering them have squares outside of the considered window. Unlabeled squares are ignored as they do not belong to $\mathcal{B}$. As long as the procedure is allowed (with penalties) to cover squares labeled ‘C’, it always returns a feasible solution (e.g., the original tiling). The running time of Procedure Retile depends on the size of the retiling window determined by d; it could be chosen small enough such that the resulting tiling subproblem is solvable in a fast manner using exact algorithms.

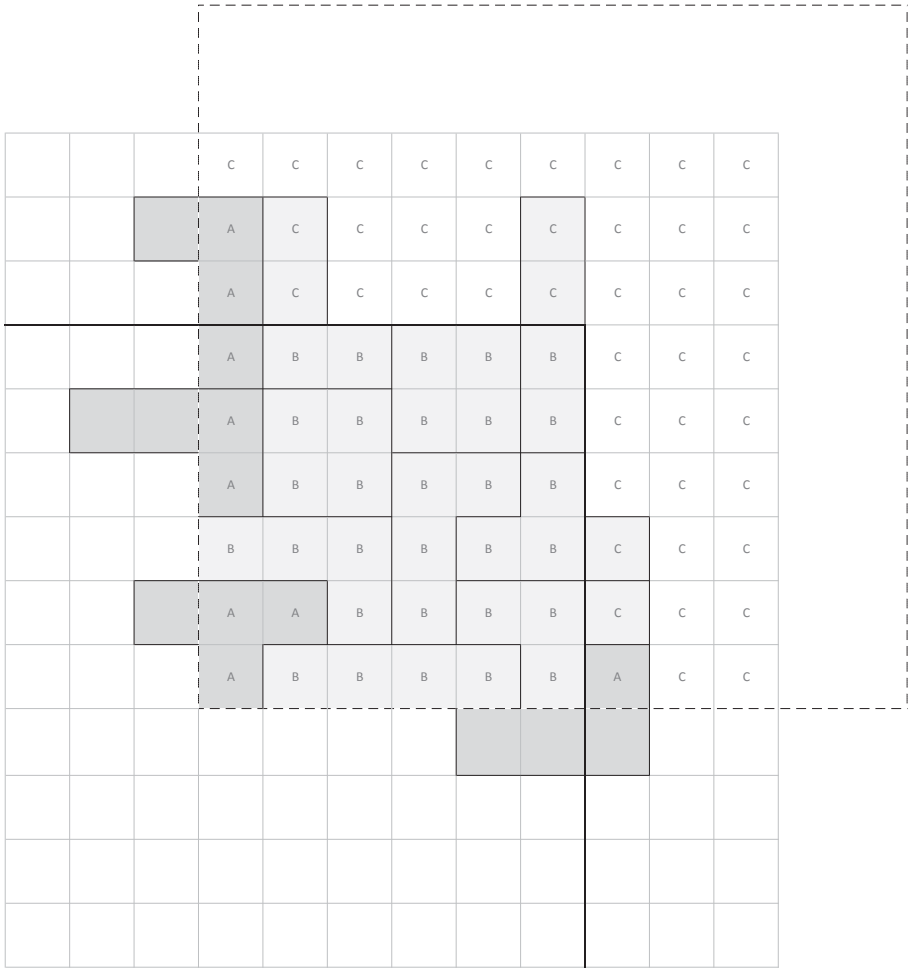


Figure 13: Initialization in Procedure Retile: A→ do not cover, B→ retile, and C→ penalize.

Procedure Retile

Input: (i) $m \times n$ master board $\mathcal{B} = \mathcal{B}_c \cup \mathcal{B}_f$ and its tiling $\mathcal{T}$ using $\mathcal{P}$, (ii) $(2d+1) \times (2d+1)$ window w to be retiled and its center (r, c) on $\mathcal{B}$, (iii) Restriction on frame: (a) pack, (b) penalize & pack, (c) do not cover, and (iv) a penalty coefficient α.

begin Initialize

  $\mathcal{B}^w = \{(i,j) : (r-d, c-d) \leq (i,j) \leq (r+d, c+d); (0,0) \leq (i,j) < (m,n)\}$

  $T_c^w = \{(k,p,q) \in \mathcal{T} \mid f_{pq}^k \subseteq \mathcal{B}^w\}$

  $T_f^w = \{(k,p,q) \in \mathcal{T} \mid 0 < |f_{pq}^k \cap \mathcal{B}^w| < |f_{pq}^k|\}$

  $B_c^w = \{(i,j) : (i,j) \in f_{pq}^k, (k,p,q) \in T_c^w; (i,j) \in \mathcal{B}_c\}$ `// center - Figure 13`

  $\mathcal{B}_{f1}^w = \{(i,j) : (i,j) \in f_{pq}^k, (k,p,q) \in T_f^w; (i,j) \in \mathcal{B}^w\}$ `// free frame`

  $\mathcal{B}_{f2}^w = \mathcal{B}^w \setminus (\mathcal{B}_c^w \cup \mathcal{B}_{f1}^w)$ `// fixed frame`

Rewrite the partitioning constraints in the model $\mathrm{P_L}$ for the tiling problem $(\mathcal{B}^w, \mathcal{P})$ as follows:

$\mathcal{B}_c^w$: $\sum_{(kpq) \in I_{ij}} x_{pq}^k = 1$, $\forall (i,j) \in \mathcal{B}_c^w$,

$\mathcal{B}_{f1}^w$: $\sum_{(kpq) \in I_{ij}} x_{pq}^k = 0$, $\forall (i,j) \in \mathcal{B}_{f1}^w$,

$\mathcal{B}_{f2}^w$: $\sum_{(kpq) \in I_{ij}} x_{pq}^k \leq 1$, $\forall (i,j) \in \mathcal{B}_{f2}^w$ **if** 'pack',

$\sum_{(kpq) \in I_{ij}} x_{pq}^k \leq p_{ij}$, $p_{ij} \leq 1$, $\forall (i,j) \in \mathcal{B}_{f2}^w$ **if** 'penalize & pack',

$\sum_{(kpq) \in I_{ij}} x_{pq}^k = 0$, $\forall (i,j) \in \mathcal{B}_{f2}^w$ **if** 'do not cover'.

if *'penalize & pack'* **then**

  Add $-\alpha \sum_{(ij) \in \mathcal{B}_{f2}^w} p_{ij}$ to the objective function of $\mathrm{P_L}$, where α is a penalty coefficient.

Solve resulting formulation using one of the exact methods.

if *'do not cover'* **and** *formulation is infeasible* **then**

  return *Infeasible*

else

  Let $\mathcal{T}^w$ be the solution tiling obtained.

  $\mathcal{T} \leftarrow (\mathcal{T} \setminus T_c^w) \cup \mathcal{T}^w$.

return $\mathcal{T}$

Randomize: This procedure attempts to improve the objective function value of a given large-scale solution by resolving smaller size tiling problems. Specifically, it reapplies Procedure Retile traversing along rows and columns of $\mathcal{B}$ with step sizes Δr and Δc, respectively. Procedure Randomize has a linear time complexity in the size of the board. Figure 14 demonstrates the procedure on an octomino meta-rectangle tiling obtained by MrTA. Figure 14c shows the 120×160 tiling in Figure 14b "randomized" through its middle. Figure 14d shows the distribution of the centers of gravity for the tiling in Figure 14c.

Procedure Randomize

Input: (i) $m \times n$ board $\mathcal{B}$ and its tiling $\mathcal{T}$ using $\mathcal{P}$, (ii) retiling window size $(2d+1) \times (2d+1)$, (iii) the vertical and horizontal increments Δr and Δc, and (iv) penalty coefficient α.

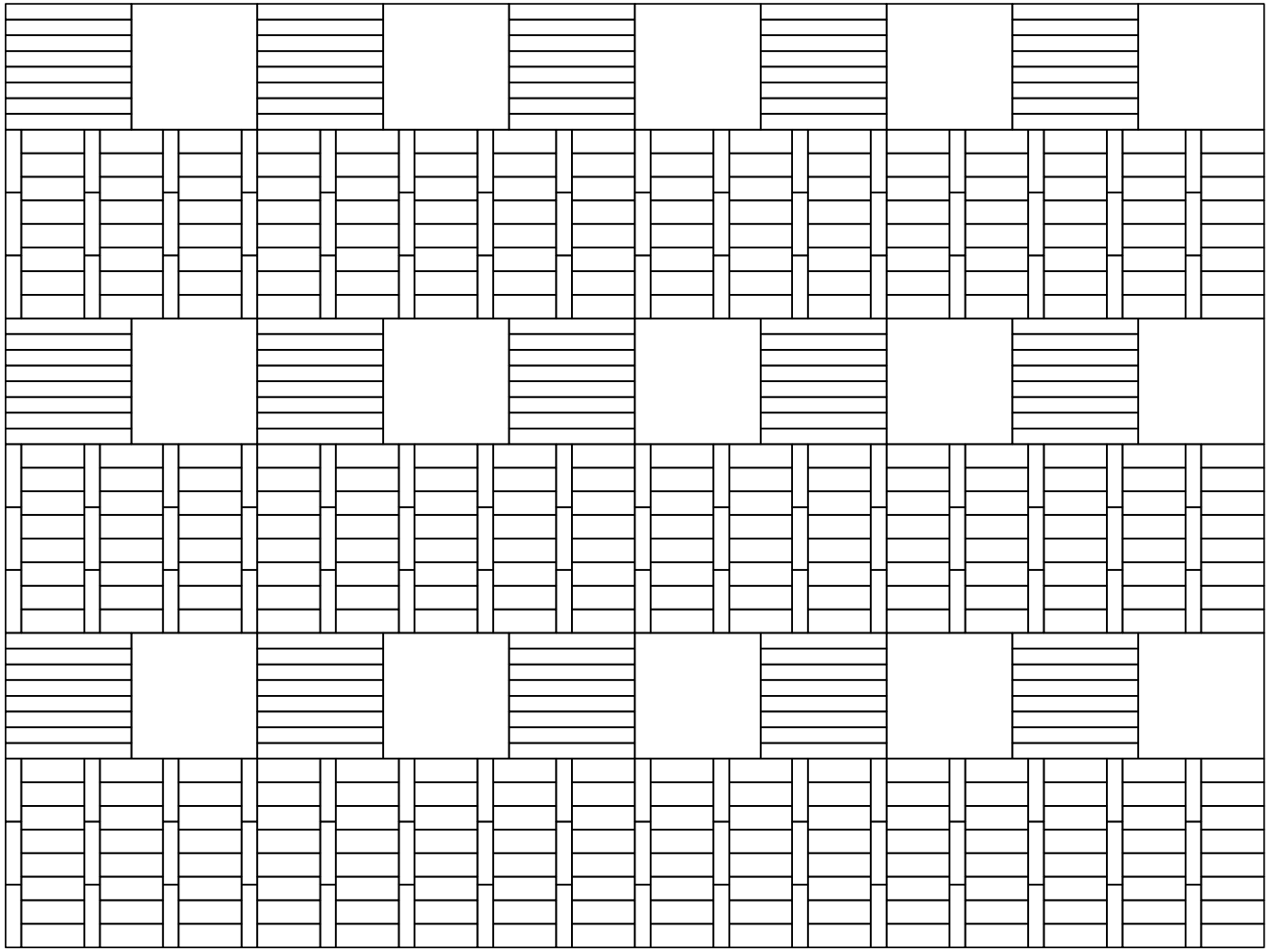
if $\mathcal{T}$ *is perfect* **then**
  $FrameType =$ 'do not cover'
else
  $FrameType =$ 'pack'
Set $r = 0$ and $c = 0$.
while $r < m$ **do**
  while $c < n$ **do**
    $\mathcal{T} \leftarrow$`Retile(` $\mathcal{B}$, $(\mathcal{T}, \mathcal{P})$, $(2d+1) \times (2d+1)$, (r, c), $FrameType$, α`)`
    $c \leftarrow c + \Delta c$
  $r \leftarrow r + \Delta r$
return $\mathcal{T}$

Smoothen: Due to the inherent computational complexity of the considered problem, construction algorithms presented in Section 2.4.1 do not necessarily obtain exact tilings for any board $\mathcal{B}$. Using construction heuristics, we still can obtain a tiling larger than the requested one and then drop some of the polyominoes along the edges to obtain an inexact tiling of $\mathcal{B}$. This procedure then traverses along the boundaries of $\mathcal{B}$ and iteratively reapplies Procedure Retile penalizing for covered members of $\mathcal{B}_f$, the frame for $\mathcal{B}$.

Procedure Smoothen

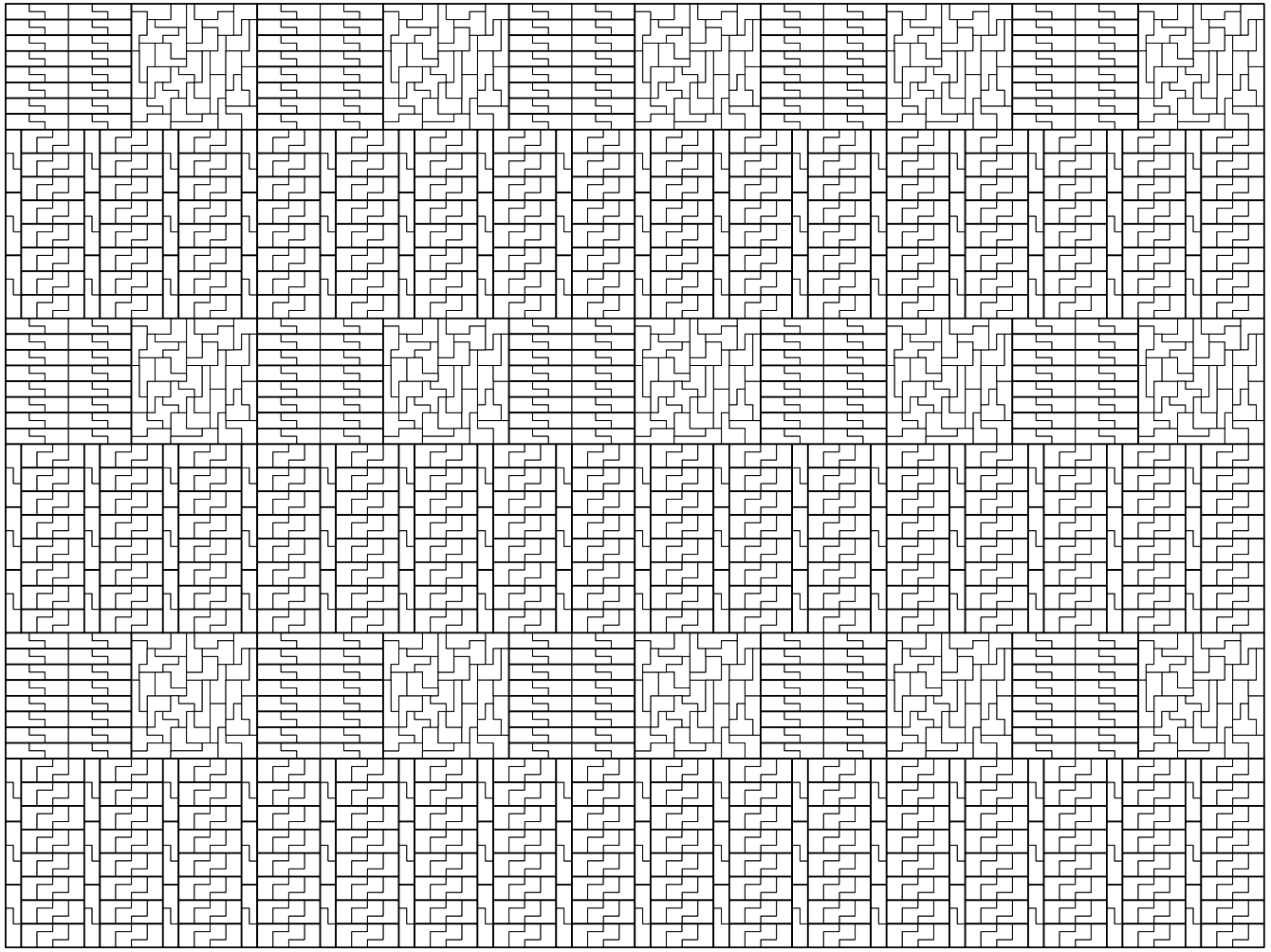
Input: (i) $m \times n$ board $\mathcal{B}$ and its imperfect tiling $\mathcal{T}$ using $\mathcal{P}$, (ii) retiling window size $(2d+1) \times (2d+1)$, (iii) penalty coefficient α, (iv) iteration limit $MaxIter$, and (v)the vertical and horizontal increments Δr and Δc.

```
1  iter = 0
2  while iter < MaxIter do
3      r = 0, c = 0
4      while c < n do
5          T ←Retile( B, (T, P), (2d + 1) × (2d + 1), (r, c), 'penalize & pack', α)
6          c ← c + Δc
7      c = n − 1
8      while r < m do
9          T ←Retile( B, (T, P), (2d + 1) × (2d + 1), (r, c), 'penalize & pack', α)
10         r ← r + Δr
11     r = m − 1
12     while r ≥ 0 do
13         T ←Retile( B, (T, P), (2d + 1) × (2d + 1), (r, c), 'penalize & pack', α)
14         r ← r − Δr
15     r = 0
16     while c ≥ 0 do
17         T ←Retile( B, (T, P), (2d + 1) × (2d + 1), (r, c), 'penalize & pack', α)
18         c ← c − Δc
19     if T is perfect then
20         return T
21     iter ← iter + 1
22 return T
```



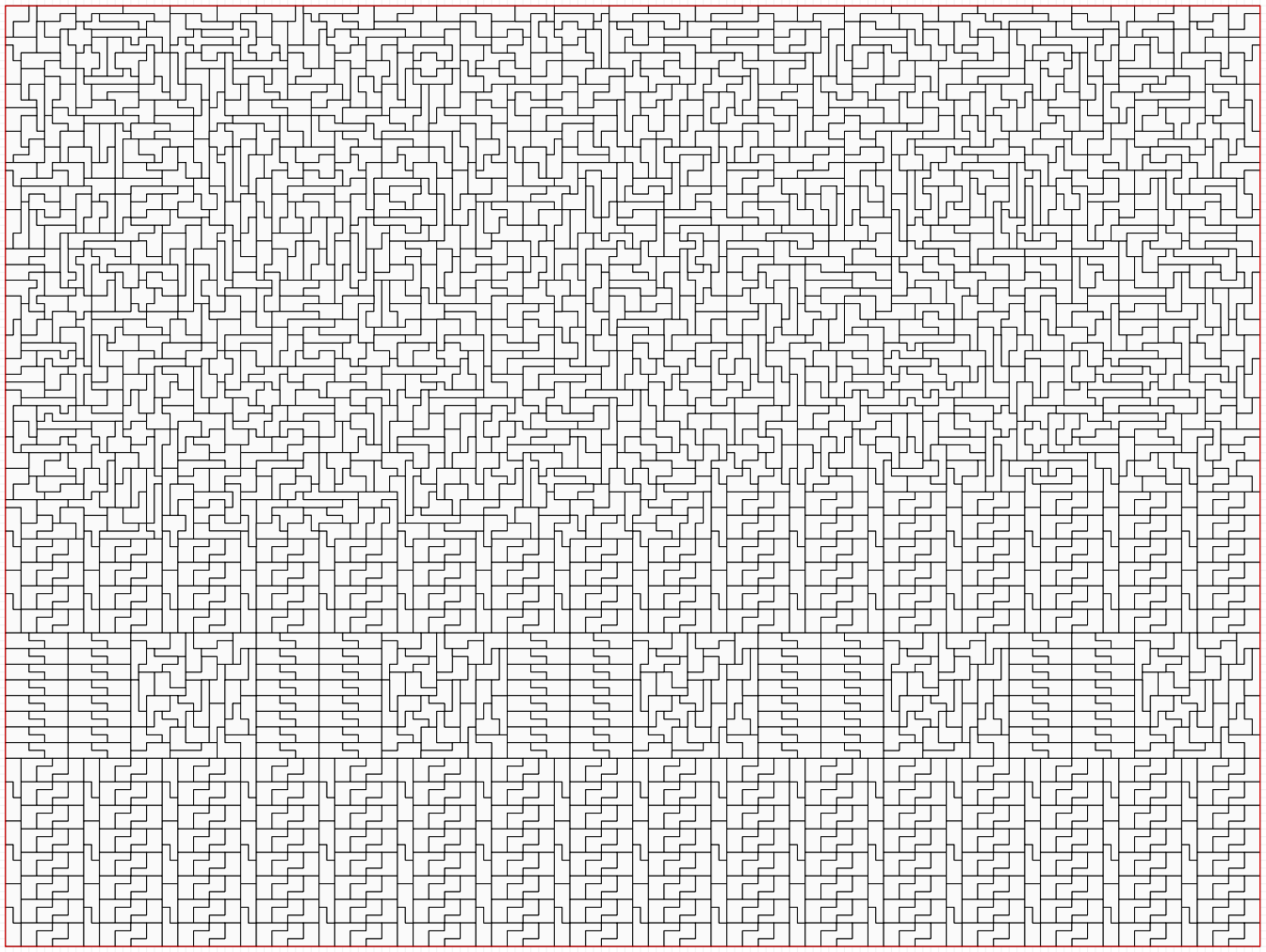
(a) 120×160 meta-rectangle.

Figure 14: Randomizing an octomino meta-rectangle tiling.



(b) 120×160 octomino tiling of the meta-rectangle.

Figure 14: (continued).



(c) Randomization in progress; the tiling in part (b) randomized through to its middle.

Figure 14: (continued).

(d) Centers of gravity for part (c).

Figure 14: (continued).

2.5 APPROXIMATION BOUNDS

In this section we show that a special case of MrTA heuristic is an approximation algorithm if the initial set of rectangles includes exactly one rectangle and no rotations are allowed. Observe that this special case of MrTA, henceforth referred to as *Paste-Side-by-Side Algorithm* (PSA), reduces to merging multiple copies of the given rectangle horizontally and vertically alongside each other (side-by-side) to create larger rectangles.

Consider $m \times n$ perfect tiling $\mathcal{T}_0$. Let r_i and c_j be the numbers of centers of gravity on row i and column j, respectively. Recall that $\sum r_i = \sum c_j = T$. The entropy of $\mathcal{T}_0$ is

$$E_0 = -\sum_{i=1}^{m} \frac{r_i}{2T} \lg\left(\frac{r_i}{2T}\right) - \sum_{j=1}^{n} \frac{c_j}{2T} \lg\left(\frac{c_j}{2T}\right) \ .$$

Assume that $\mathcal{T}_1$ is obtained from $\mathcal{T}_0$ by "pasting" it side-by-side a times vertically and b times horizontally. Tiling $\mathcal{T}_1$ has $a\,m$ rows and $b\,n$ columns. Let r_{ij} be the number of centers of gravity on row ij for $1 \le i \le a$ and $1 \le j \le m$. Similarly, let c_{ij} be the number of centers of gravity on column ij for $1 \le i \le b$ and $1 \le j \le n$. Then for all i and j:

$$r_{ij} = b\ r_j \quad \text{and} \quad c_{ij} = a\ c_j.$$

Then entropy of $\mathcal{T}_1$ is obtained as follows:

$$
\begin{aligned}
E_1 &= -\sum_{i=1}^{a}\sum_{j=1}^{m} \frac{r_{ij}}{2abT} \lg\left(\frac{r_{ij}}{2abT}\right) - \sum_{i=1}^{b}\sum_{j=1}^{n} \frac{c_{ij}}{2abT} \lg\left(\frac{c_{ij}}{2abT}\right) \\
&= -\sum_{i=1}^{a}\sum_{j=1}^{m} \frac{b\ r_j}{2abT} \lg\left(\frac{b\ r_j}{2abT}\right) - \sum_{i=1}^{b}\sum_{j=1}^{n} \frac{a\ c_j}{2abT} \lg\left(\frac{a\ c_j}{2abT}\right) \\
&= -\sum_{i=1}^{a}\sum_{j=1}^{m} \frac{r_j}{2aT} \lg\left(\frac{r_j}{2aT}\right) - \sum_{i=1}^{b}\sum_{j=1}^{n} \frac{c_j}{2bT} \lg\left(\frac{c_j}{2bT}\right) \\
&= -\sum_{j=1}^{m} a\frac{r_j}{2aT} \lg\left(\frac{r_j}{2aT}\right) - \sum_{j=1}^{n} b\frac{c_j}{2bT} \lg\left(\frac{c_j}{2bT}\right) \\
&= -\sum_{j=1}^{m} \frac{r_j}{2T} \lg\left(\frac{r_j}{2aT}\right) - \sum_{j=1}^{n} \frac{c_j}{2T} \lg\left(\frac{c_j}{2bT}\right) \\
&= -\sum_{j=1}^{m} \frac{r_j}{2T} \left(\lg\left(\frac{r_j}{2T}\right) + \lg\left(\frac{1}{a}\right)\right) - \sum_{j=1}^{n} \frac{c_j}{2T} \left(\lg\left(\frac{c_j}{2T}\right) + \lg\left(\frac{1}{b}\right)\right) \\
&= -\left(\sum_{j=1}^{m} \frac{r_j}{2T} \lg\left(\frac{r_j}{2T}\right) + \sum_{j=1}^{m} \frac{r_j}{2T} \lg\left(\frac{1}{a}\right)\right) - \left(\sum_{j=1}^{n} \frac{c_j}{2T} \lg\left(\frac{c_j}{2T}\right) + \sum_{j=1}^{n} \frac{c_j}{2T} \lg\left(\frac{1}{b}\right)\right) \\
&= -\left(\sum_{j=1}^{m} \frac{r_j}{2T} \lg\left(\frac{r_j}{2T}\right) + \sum_{j=1}^{n} \frac{c_j}{2T} \lg\left(\frac{c_j}{2T}\right)\right) - \left(\sum_{j=1}^{m} \frac{r_j}{2T} \lg\left(\frac{1}{a}\right) + \sum_{j=1}^{n} \frac{c_j}{2T} \lg\left(\frac{1}{b}\right)\right) \\
&= E_0 - \frac{T}{2T} \lg\left(\frac{1}{a}\right) - \frac{T}{2T} \lg\left(\frac{1}{b}\right) \\
&= E_0 + \lg\sqrt{a\ b}\,.
\end{aligned}
$$

Since the theoretical upper bound for entropy of any perfect tiling of an $a\,m \times b\,n$ board is $\lg(a\ m + b\ n)$, the relative optimality gap for $\mathcal{T}_1$ is bounded by:

$$
1 - \frac{E_0 + \lg\sqrt{a\,b}}{\lg(a\,m + b\,n)}\,. \tag{2.15}
$$

Proposition 8. *PSA is asymptotically optimal.*

Proof. We show that

$$
\lim_{(a,b)\to(\infty,\infty)} f(a,b) = \lim_{(a,b)\to(\infty,\infty)} \frac{E_0 + \lg\sqrt{a\,b}}{\lg(a\,m + b\,n)} = 1\,.
$$

Consider transformation to polar coordinates:

$$a = r\,cos\,\theta \quad \text{and} \quad b = r\,sin\,\theta\,.$$

Then $f(a,b)$ can be rewritten as

$$\Gamma(r,\theta) = \frac{E_0 + \frac{1}{2}\lg(r^2\,\sin\,\theta\,\cos\,\theta)}{\lg(m\,r\,\cos\,\theta + n\,r\,\sin\,\theta)}\,.$$

Note that $\left\{\theta \,|\, \theta = k\,\frac{\pi}{2}\,,\, k \in \mathbb{Z}\right\}$ is not in the domain of $\Gamma(r,\theta)$ because $a \geq 1$ and $b \geq 1$. Then it is easy to see that

$$r \,\to\, \infty \quad \Leftrightarrow \quad (a,b) \,\to\, (\infty,\infty)\,.$$

Consequently,

$$\begin{aligned}
\lim_{(a,b)\to(\infty,\infty)} f(a,b) &= \lim_{r\to\infty} \Gamma(r,\theta) \\
&= \lim_{r\to\infty} \frac{E_0 + \frac{1}{2}\lg(r^2\,\sin\,\theta\,\cos\,\theta)}{\lg(m\,r\,\cos\,\theta + n\,r\,\sin\,\theta)} \\
&= \frac{+\infty}{+\infty} \quad \text{(apply L'Hospital's rule)} \\
&= \lim_{r\to\infty} \frac{r\,(m\,\cos\,\theta + n\,\sin\,\theta)}{m\,\cos\,\theta + n\,\sin\,\theta} \frac{2r\,\sin\,\theta\,\cos\,\theta}{2r^2\,\sin\,\theta\,\cos\,\theta} \\
&= 1\,.
\end{aligned}$$

Since the limit is independent of the angle θ, the direction one approaches to the limiting value, limit of $f(a,b)$ as $(a,b) \to (\infty,\infty)$ exists and is equal to 1. Hence, the limiting value of the bound (2.15) is 0 and the result follows. □

Proposition 9. *If the entropy of the initial tiling $\mathcal{T}_0$ is at least $(\frac{1}{2}+\epsilon)\lg(m+n)$, $\epsilon \in [0,\frac{1}{2}]$, then PSA has an approximation guarantee of*

$$\frac{1}{2}\frac{\lg(a\,b\,m\,+\,a\,b\,n)}{\lg(a\,m\,+\,b\,n)} + \epsilon\frac{\lg(m\,+\,n)}{\lg(a\,m\,+\,b\,n)} \;\geq\; \frac{1}{2}\,.$$

Proof. Recall that $\lg(a\,m\ +\ b\,n)$ is an upper bound for an optimal solution of our problem. Then

$$\begin{aligned}\frac{E_0\ +\frac{1}{2}\lg(ab)}{\lg(a\,m\ +\ b\,n)} &\geq \frac{(\frac{1}{2}\ +\ \epsilon)\lg(m\ +\ n)\ +\frac{1}{2}\lg(a\,b)}{\lg(a\,m\ +\ b\,n)}\\ &= \frac{1}{2}\frac{\lg(a\,b\,m\ +\ a\,b\,n)}{\lg(a\,m\ +\ b\,n)} + \epsilon\frac{\lg(m\ +\ n)}{\lg(a\,m\ +\ b\,n)}\\ &\geq \frac{1}{2}\frac{\lg(a\,m\ +\ b\,n)}{\lg(a\,m\ +\ b\,n)} = \frac{1}{2}\,.\end{aligned}$$

□

It is important to note that the theoretical upper bound of $\log(m+n)$ is hard to achieve when $m \neq n$. It follows from the fact that either rows or columns, whichever is lesser in number, needs to accommodate more centers of gravity per row or column as $\sum r_i = \sum c_j = T$.

Proposition 10. *If $a = b$ and the initial solution $\mathcal{T}_0$ is ϵ-optimal with respect to the theoretical upper bound, then the solution obtained by PSA is also ϵ-optimal.*

Proof.

$$\begin{aligned}\frac{E_0\ +\frac{1}{2}\lg(ab)}{\lg(a\,m\ +\ b\,n)} &\geq \frac{(1-\epsilon)\lg(m\ +\ n)\ +\frac{1}{2}\lg(a^2)}{\lg(a\,(m\ +\ n))}\\ &= \frac{\lg(a(m\ +\ n))}{\lg(a\,(m\ +\ n))} - \epsilon\frac{\lg(m+n)}{\lg(a(m+n))}\\ &= 1 - \epsilon\frac{\lg(m+n)}{\lg(a(m+n))} \geq 1-\epsilon\,.\end{aligned}$$

□

Corollary 1. *If $a = b$ and the initial solution achieves the theoretical upper bound $\log(m+n)$, then PSA always returns an optimal solution.*

2.6 COMPUTATIONAL RESULTS

In this section, we present a computational study to evaluate performance of the exact and heuristic approaches proposed in this chapter. The first part aims at assessing the performance of the branching rule and delayed column generation approaches described in Section 2.3. The second part summarizes results for heuristic algorithms and procedures described in Section 2.4.

All routines except branch-and-price (B&P) are coded in PYTHON; CPLEX PYTHON API [74] is used for routines requiring CPLEX (e.g., CPLEX with the proposed branching strategy). The B&P algorithm is coded in C++ and uses the open source COIN-OR BCP framework [37] for UNIX environment. CLP is used as LP solver for BCP. All code is single-threaded. A 32-bit Windows 7 machine with dual-core Intel Xeon E3110 @ 3.00 GHz processor and 3 GB RAM is used for all computations. Cygwin 1.7.9 on the same machine is used for BCP.

2.6.1 Results for Exact Approaches

We compare three solution approaches to solve the model P_L: (i) CPLEX, (ii) CPLEX using the branching strategy (CPLEX-BR) proposed in Section 2.3, and (iii) branch-and-price using BCP. For CPLEX and CPLEX-BR, all cuts and heuristics are enabled. The BCP implementation is a pure B&P algorithm (i.e., no cuts or heuristics are used). Besides running times, the number of nodes created/processed by each algorithm is also reported to demonstrate the search capability of the algorithms. Time limit of 604,800 seconds (1 week) is enforced for all experiments.

We consider tetromino, pentomino, and octomino families. Specifically, all members of tetrominoes and pentominoes are used (see Figure 1). However, this is not possible for octominoes as $|\mathcal{F}_8| = 369$ and this number grows to $2,725$ if we include rotations and reflections; the resulting formulations are far beyond tractable even for small board dimensions. Therefore, a subset of $\mathcal{F}_8$ with only 13 octominoes is selected for the experiments. For this subset of octominoes, illustrated in Figure 15, we have $|\mathcal{P}| = 76$.

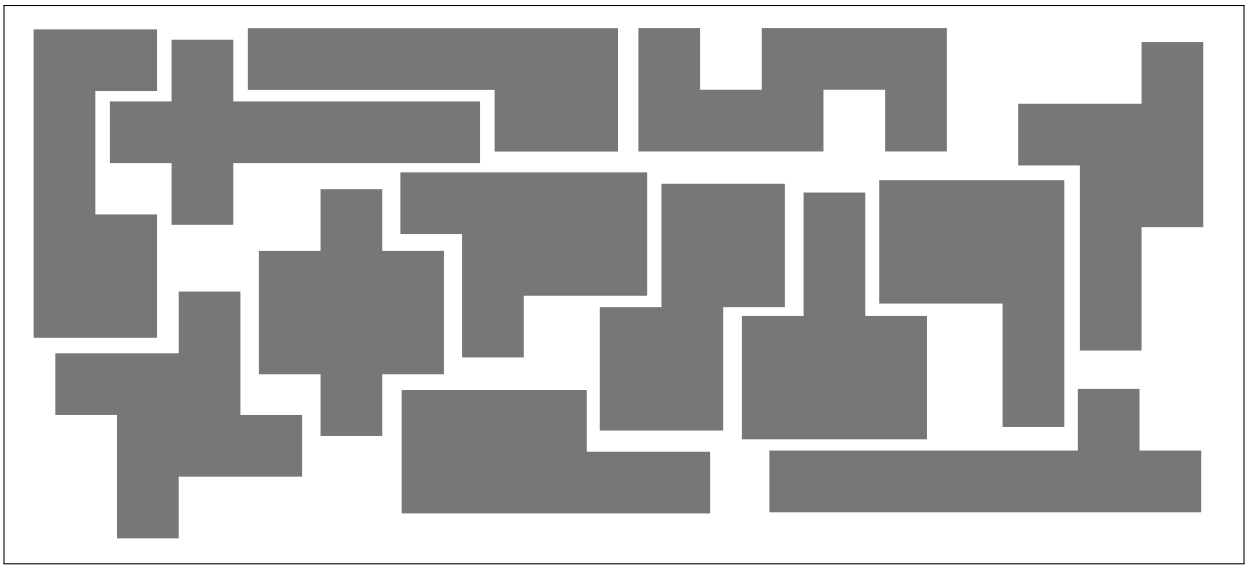


Figure 15: The subset of octomino family, $\mathcal{F}_8$, used in experiments.

A termination criterion of 0.5% relative gap is used for tetrominoes and pentominoes. For octominoes, this cut-off percentage gap is raised to 3.0%, since this family is computationally challenging for the LP solvers because of denser constraint matrices. The performance of the algorithms is evaluated for the case where finding a feasible solution is trivial (e.g., a 25×25 board for pentominoes that could be tiled using only 'l' pentomino) and the other cases where a feasible solution may not be trivial (e.g., a 22×26 board for tetrominoes). Large board dimensions are selected rather straight forward as CPLEX and CPLEX-BR could not get beyond moderate board sizes in the experiments.

Table 1 summarizes the results for the tetromino family. CPLEX and CPLEX-BR are better than BCP up to dimension 20×20, however, beyond this size BCP dominates both algorithms. Neither CPLEX nor CPLEX-BR are capable of solving instances beyond the 32×32 board dimension. Observe that CPLEX has an unstable search tree size relative to the other two algorithms. For small dimensions, CPLEX finds good solutions at the root node using heuristics. Beyond small board dimensions, CPLEX fails as search space becomes intractable very fast. CPLEX-BR has a better search performance, however, for larger dimensions even solving LPs becomes difficult. CPLEX-BR outperforms CPLEX in

almost all instances. BCP has a rather steady increase in the search tree size and running time. Though it does not use any heuristics, delayed column generation approach gives it the ability to process nodes much faster. The problem becomes intractable beyond 60×80 board size.

Table 1: Computational results for tetromino family: exact approaches.

Size		Nodes Created/Processed						Running Time (secs)		
		CPLEX		CPLEX-BR		BCP				
# Rows	# Cols	Created	Processed	Created	Processed	Created	Processed	CPLEX	CPLEX-BR	BCP
12	12	10	10	93	60	1,153	673	1	5	7
12	18	99	50	30	30	275	144	5	8	4
12	24	888	583	694	460	1,875	1,103	85	75	41
16	16	1	1	1	1	235	118	5	5	5
16	19	1	1	1	1	2,265	1,303	9	9	62
16	20	1	1	1	1	309	161	8	7	9
16	25	1	1	1	1	1,297	670	44	46	44
20	20	1	1	1	1	977	496	29	30	219
20	23	1	1	1	1	497	250	56	56	28
20	24	794	499	651	330	539	270	467	222	32
20	30	1	1	1	1	805	408	114	115	58
22	26	117,552	76,912	479	240	2,429	1,235	16,635	298	132
24	24	530	530	907	464	789	400	711	560	58
24	25	17,998	10,109	1,859	960	715	362	7,125	1,386	55
25	28	831	509	1,131	580	1,151	585	1,612	1,181	97
28	30	†	†	18,965	9,600	1,745	902	†	38,708	197
30	30			3,444	2,884	1,731	880		13,567	198
32	32			1,309	660	3,075	1,556		4,960	510
40	40			†	†	4,259	2,165		†	1,198
40	60					39,005	19,707			18,349
40	80					49,587	25,085			49,477
50	80					91,641	46,276			106,292
60	80					194,041	97,827			496,570

† No feasible solution obtained beyond this size in 604,800 seconds (1 week) time limit.

Most of the conclusions drawn for the tetromino family are also valid for the pentomino family, see results in Table 2. Though CPLEX cannot find a solution for 23×25 board in one week, BCP is able to find a solution within allowed tolerances in less than four minutes.

Table 2: Computational results for pentomino family: exact approaches.

Size		Nodes Created/Processed						Running Time (secs)		
		CPLEX		CPLEX-BR		BCP				
# Rows	# Cols	Created	Processed	Created	Processed	Created	Processed	CPLEX	CPLEX-BR	BCP
10	10	1	1	1	1	493	290	1	1	2
10	13	1	1	1	1	381	238	2	2	3
13	15	138	80	101	50	1,597	956	13	29	24
15	15	1	1	1	1	1,529	861	13	13	34
15	17	1	1	1	1	2,159	1,154	22	22	60
15	19	1	1	1	1	591	307	16	16	19
17	20	1	1	1	1	719	363	71	75	33
20	20	1	1	1	1	463	231	68	66	28
20	24	1	1	1	1	1,297	659	623	616	100
20	30	1	1	1	1	1,503	780	328	325	141
23	25	†	†	653	340	1,977	998	†	3,005	205
25	25			499	260	3,543	1,836		2,353	400
25	29			1,101	559	2,213	1,117		5,276	396
30	30			248,955	248,400	4,015	2,029		61,909	802
40	40			†	†	7,875	3,954		†	4,553
40	60					8,015	4,061			11,789
40	80					138,577	69,859			276,085
40	100					241,883	122,193			435,134
50	100					155,641	78,301			508,267
60	80					70,243	35,244			320,726
60	100					128,255	64,446			441,318

† No feasible solution obtained beyond this size in 604,800 seconds (1 week) time limit.

As the number of squares that each octomino covers (i.e., 8) is large, the constraint matrix for the related formulation is denser. Hence, the LP solution times are larger for all algorithms and the time spent per tree node increases sharply as it can be observed from Table 3. Both CPLEX and CPLEX-BR fail at rather small board sizes. BCP is more stable and shows almost the same performance as for tetrominoes and pentominoes in terms of tree search.

In summary, we conclude that the branching strategy and the B&P implementation based on the delayed column generation dominates direct use of an MIP solver with traditional

Table 3: Computational results for octomino family: exact approaches.

Size		Nodes Created/Processed						Running Time (secs)		
		CPLEX		CPLEX-BR		BCP				
# Rows	# Cols	Created	Processed	Created	Processed	Created	Processed	CPLEX	CPLEX-BR	BCP
12	12	1	1	1	1	1,391	801	11	11	23
12	16	2,247	1,573	149	80	2,289	1,305	272	114	65
12	20	476	476	143	80	3,613	2,068	436	120	150
16	16	79	40	759	440	453	250	71	583	33
20	20	734	600	883	527	1,975	1,013	2,812	2,900	265
20	24	279	169	11,290	9,264	1,027	526	1,121	32,157	160
20	28	3,606	2,498	39,591	34,688	4,577	2,426	44,466	294,453	995
24	24	5,680	4,032	10,033	7,724	571	295	55,989	66,508	194
24	28	5,257	3,210	2,330	1,392	2,545	105	89,428	30,456	375
28	28	†	†	6,506	4,769	4,229	2,160	†	201,501	1,739
32	32			10,117	5,937	41,633	21,662		514,897	20,452
40	40			†	†	10,597	5,409		†	21,882
48	48					163,655	84,122			330,999

† No feasible solution obtained beyond this size in 604,800 seconds (1 week) time limit.

branching. The major aim of our efforts is to have an exact method that can solve moderate-size problems. These solutions are used for construction/improvement heuristics.

2.6.2 Results for Heuristic Approaches

A computational study of ZiA using octomino family is povided in Table 4. For different initial octomino tilings and zoom-in levels we report the setup time to construct initial tilings, the relative optimality gap after t applications of ZiA, the time spent to perform one iteration, the number of polyominoes used in the final tiling, and the final size of the board. Since each polyomino itself is tiled using entropy maximization, each application of ZiA further "randomizes" obtained tiling. This effect can be observed by comparing the relative optimality gap as t increases. ZiA is very successful as it constructs large irregular tilings of high quality in a very short time; it is able to obtain a $16\,384 \times 16\,384$ tiling that

consists of more than 33 million octominoes, with less than 0.02% optimality gap in less than one hour.

Table 4: Computational results for octomino family: ZiA.

	Initial Tiling Size:	12×12	20×24	32×32
	Zoom Level:	4×4	6×6	8×8
# Zoom-ins	Setup Time (secs)†	$23 + 2,294$	$160 + 3,075$	$20,462 + 7,763$
t=1	Relative Gap %	0.73	0.39	0.12
	Time (secs)	0.04	0.15	0.46
	# Polyominoes	288	2,160	8,192
	Final Tiling Size	48×48	120×144	256×256
t=2	Relative Gap %	0.21	0.16	0.03
	Time (secs)	0.33	3.95	26.09
	# Polyominoes	4,608	77,760	524,288
	Final Tiling Size	192×192	720×864	2,048×2,048
t=3	Relative Gap %	0.11	0.11	0.02
	Time (secs)	4.50	172	$1,145 + 1,466^{\varsigma}$
	# Polyominoes	73,728	2,799,360	33,554,432
	Final Tiling Size	768×768	4,320×5,184	16,384×16,384

† 3% relative gap and 1,800 seconds time limit enforced in finding tiling of individual polyominoes at each zoom level.

ς Due to memory limitations, data is processed sequentially: real computation time + read from / write to disk time.

Results for pentomino and octomino families using MrTA (with randomization procedure) are summarized in Table 5. A set of exactly tileable rectangles, with dimensions at most 20×20, are provided to the algorithm as input. Subsequently, different sized large tilings are generated, which are further "randomized" using one application of Procedure Randomize with two different retiling window sizes. A single pass of the procedure is able to get the optimality gap below 1% except one instance. Note that multiple passes with different window size are also possible.

Table 5: Computational results for pentomino and octomino families: MrTA.

	MrTA†				Randomize (after MrTA)‡			
	setup (secs)	size	gap %	time (secs)	retiling area	retilings / pass	gap %	time (secs)
Pentominoes	616	100 x 150	5.53	1	9 x 9	938	0.54	424
					15 x 15	306	0.52	517
		190 x 310	5.72	1	9 x 9	3,681	0.56	3,337
					15 x 15	1,202	0.54	1,615
		500 x 500	11.65	1	9 x 9	15,625	0.06	8,305
					15 x 15	5,102	0.04	12,922
Octominoes	1,484	80 x 100	4.23	1	9 x 9	500	0.62	1,174
					15 x 15	163	0.53	2,325
		220 x 360	8.22	1	9 x 9	4,950	1.89	24,580
					15 x 15	1,616	0.63	42,988
		500 x 500	3.11	1	9 x 9	15,625	0.38	31,792
					15 x 15	5,102	0.06	78,859

† $(\bar{x}, \bar{y}) \leq (20, 20)$. This have to be done only once for each polyomino family.

‡ CPLEX-BR and BCP used for retiling of pentominoes and octominoes, respectively; 4% optimality gap and 300 seconds time limit enforced.

2.7 PHASED ARRAY ANTENNA SIMULATIONS

In this section we report on our phased array antenna simulations with irregular polyomino tilings obtained using the developed optimization approach. The simulation tool is provided by the Air Force Research Laboratory (AFRL), see also [90]. The goal of the experiments is to verify that the proposed information-theoretic entropy-based concept is suitable for measuring the irregularity of a polyomino tiling and that, in fact, such irregularity contributes to antenna performance as indicated by previous results in the literature.

We consider four time delay control scenarios:

- element level control (E),
- rectangular 2×4-subarray level control (R),

- 'partially optimized' (6-8% optimality gap) polyomino-shaped subarray level control (P),
- 'completely optimized' (<0.4% optimality gap) polyomino-shaped subarray level control (P+).

Octomino family from Figure 15 is used for scenarios P and P+. For each scenario we report the results for four different array sizes: 32×32, 64×64, 128×128, and 256×256. In all simulations, (i) we use isotropic antenna elements, (ii) elements are separated by half wavelength at the center frequency f_c, (iii) antenna operates at $1.3f_c$, and (iv) antenna is steered to $(0.5, 0.5)$ in UV space. Sidelobe power values reported are normalized.

Figure 16 includes four plots demonstrating results for each scenario as the array size varies. Area gain, taper loss, and scan loss are several measures that are independent of the array structure. Power gain monotonically increases with the array size. Its highest and lowest values correspond to scenarios E and P+, respectively. However, the difference is almost negligible (around 1.5 dB). The average sidelobe levels also decrease monotonically and are about white noise level for all scenarios and array sizes. The most important characteristic is the peak sidelobe level. Figure 16 confirms that using completely optimized (in terms of the proposed irregularity metric) octomino tilings results in a significant suppression of sidelobes. The peak sidelobe is about -11dB for rectangular subarrays (R) and about -19dB for partially optimized tilings (P). The peak sidelobe level does not change much for partially optimized tilings (P) as the array size increases. However, for completely optimized tilings (P+) each quadrupling of the array size results in decrease of the peak sidelobe by approximately 5dB. This result suggests that finding near optimal solutions is important as it substantially improves the performance of the tiling. For 256×256 completely optimized array structure (P+), the peak sidelobe is close to the peak sidelobe obtained with element level control. Thus, irregularly tiled polyomino-shaped subarrays provide an attractive easy-to-implement alternative to phased array antennas with element level control.

In Figure 20, we provide contour and surface plots of radiation patterns for 128×128 array size; corresponding partially and completely optimized tilings are given in Figure 19. With element level control, beam formation is perfect as it is observed in Figures 20a and20b. With rectangular subarrays, sidelobes are almost as significant as the main beam. With a partially optimized tiling, a large set of sidelobes appears as shown in Figures 20e and 20f;

however, the peak sidelobe level is lower in comparison to the case of rectangular subarrays. Note that the radiation pattern behavior is somewhat similar to the periodicity pattern observed in the tiling in Figure 19a. With an optimally tiled array structure shown in Figure 19b, sidelobes are suppressed to the white noise level and the peak sidelobe is not significant, see Figures 20g and 20h.

Analysis for 128×128 array is also valid for 64×64 and 256×256 array sizes. Magnetic response for these array sizes are given in Figures 17 and 18 and Figures 21 and 22. The results presented in this section are very encouraging as they prove that one could obtain almost perfect beam formation by using irregularly tiled polyomino subarrays and eliminate the need to implement complex and expensive element level controls in antenna design. Redesign of antenna geometries using our results will have a significant impact considering the wide range of fields where phased array antennas are used. Furthermore, eliminating possibility for sidelobe jamming, our results have a strategic importance in electronic warfare.

2.8 CONCLUSION

Irregularly tiled polyomino-shaped subarrays improve performance of phased array antennas and in this work we focus on obtaining such tilings with integer programming/combinatorial optimization techniques. Tiling a region with a given set of polyominoes is a hard combinatorial optimization problem; deciding whether an exact tiling exists is an NP-complete problem. We model the tiling problem as an exact set covering problem, where irregularity is measured with an information-theoretic entropy-based objective function. The resulting mathematical program is nonlinear, however, it can be linearized and relaxed to a certain degree. A B&P framework with novel branching strategy and duality-based lower-bounding is proposed to solve the problem exactly. Computational results show that our approach significantly outperforms a state-of-the-art commercial solver. Successful heuristic and approximation algorithms are developed to obtain large size tilings. Our simulation study using software provided by ARFL confirms that irregularity of the obtained tilings substantially improves antenna performance.

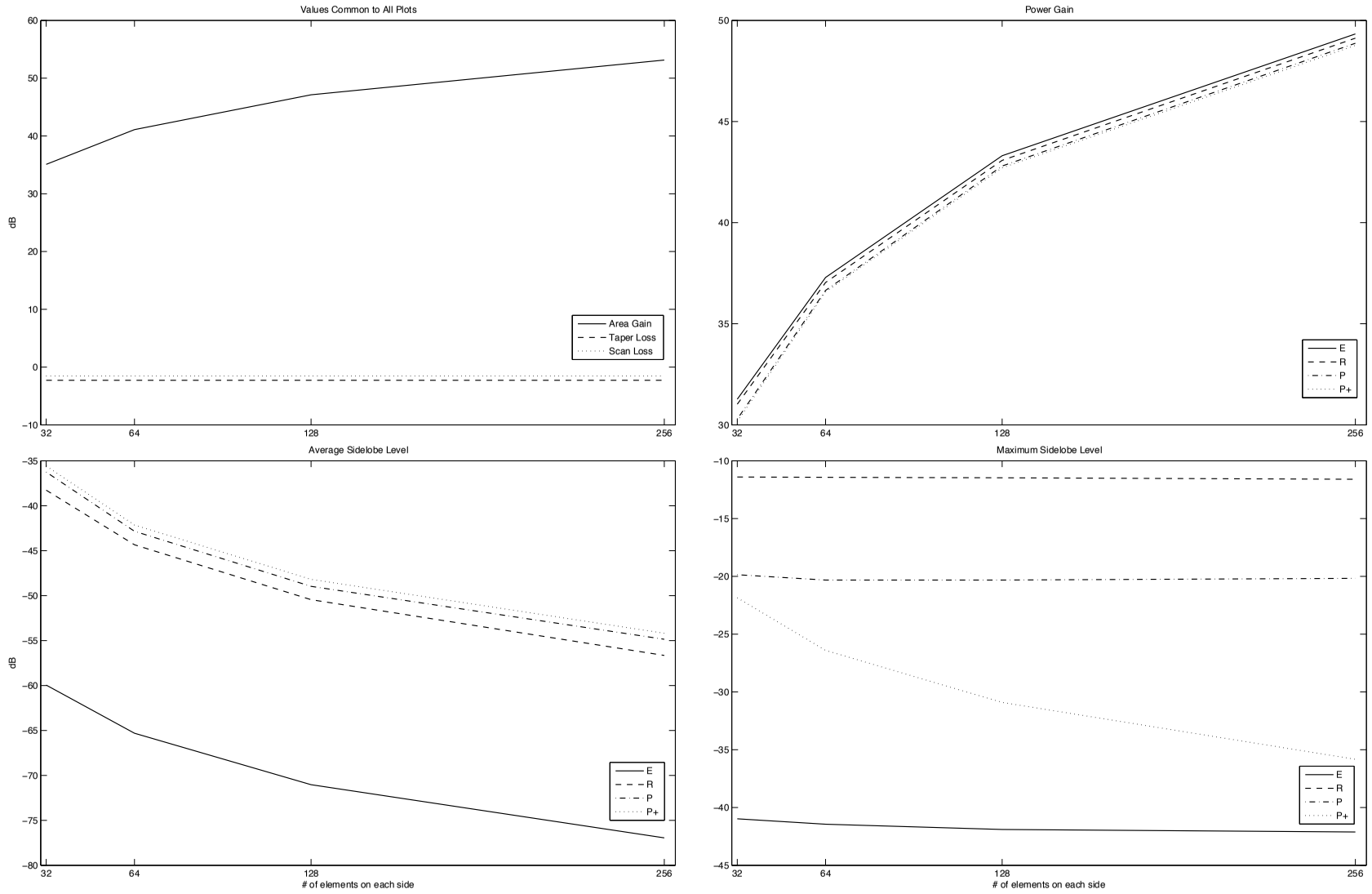
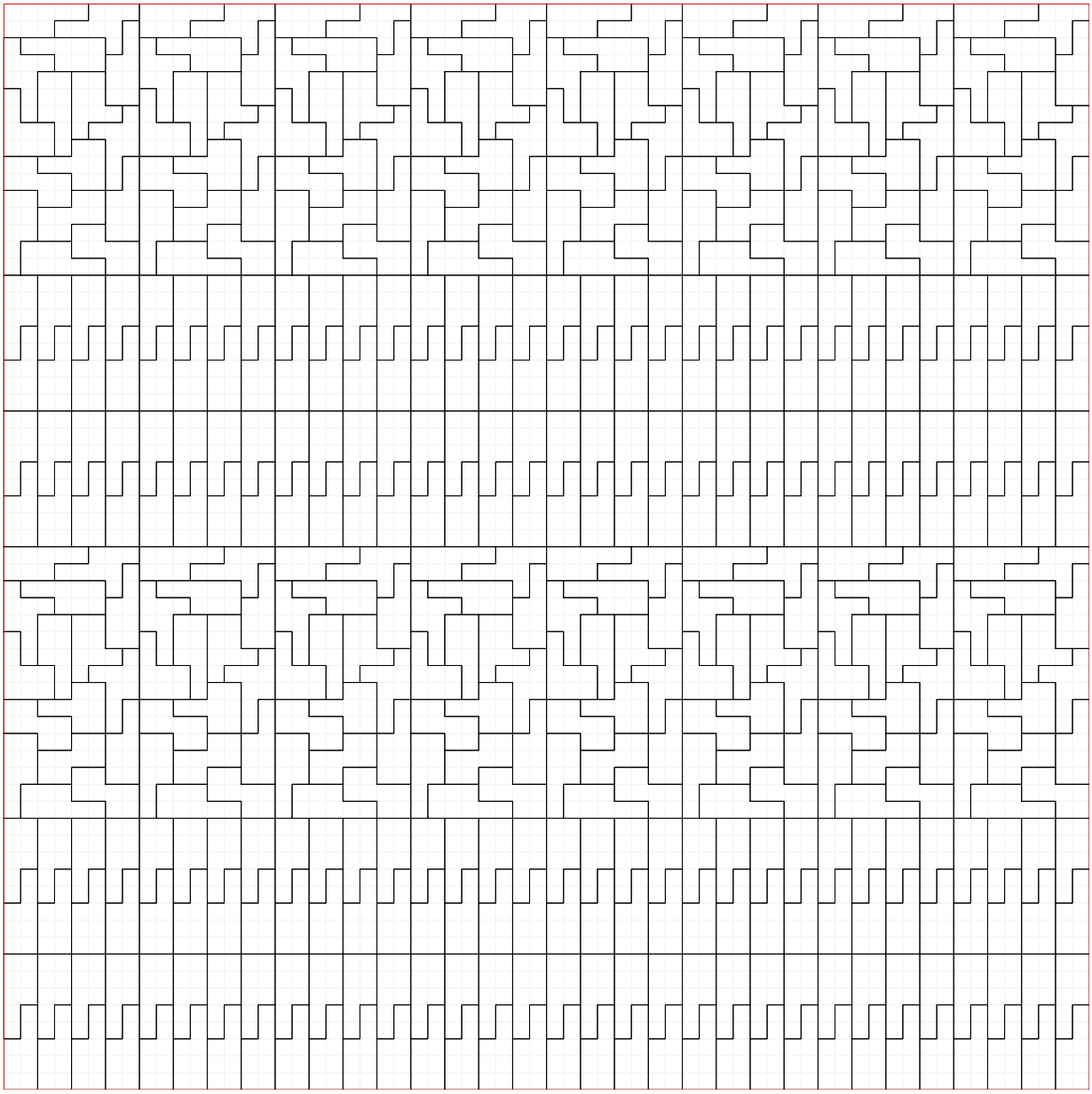
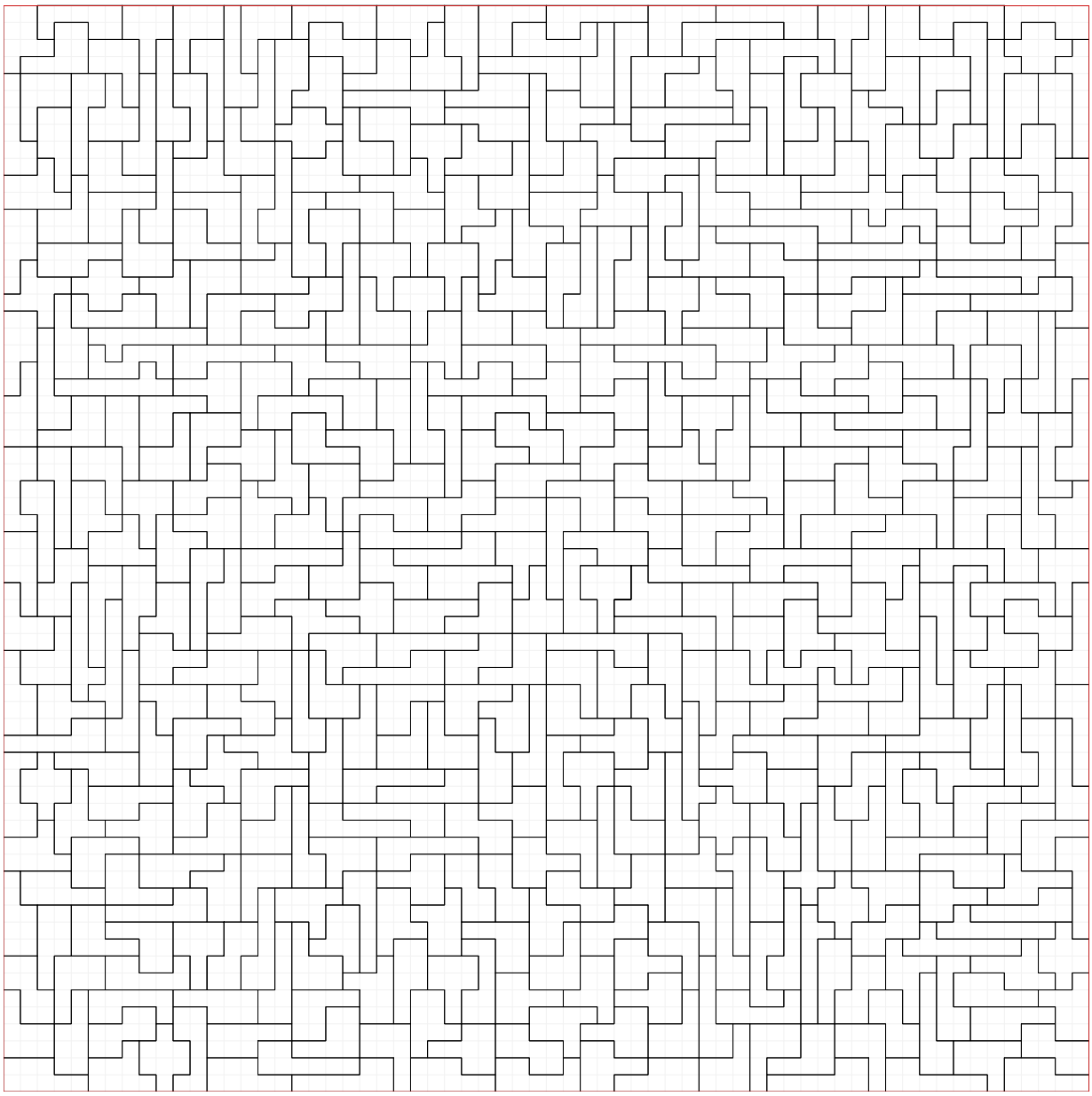


Figure 16: Summary of array antenna simulations results: time delay control at element level (**E**), 2×4 rectangular subarray level (**R**), partially optimized octomino subarray level (**P**), and completely optimized octomino subarray level (**P+**).



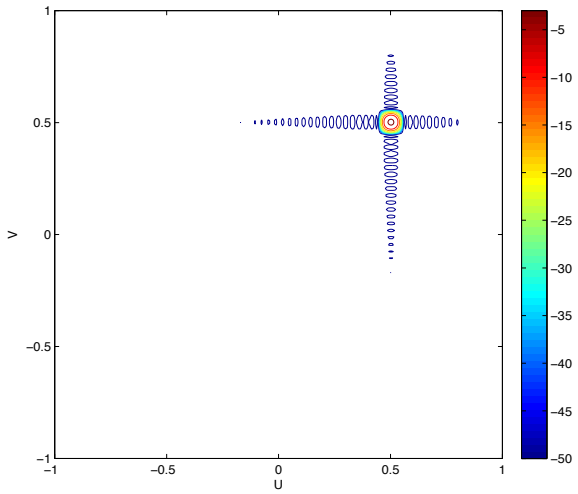
(a) P: 7.22% optimality gap

Figure 17: Partially and completely optimized polyomino tilings for 64×64 array size.

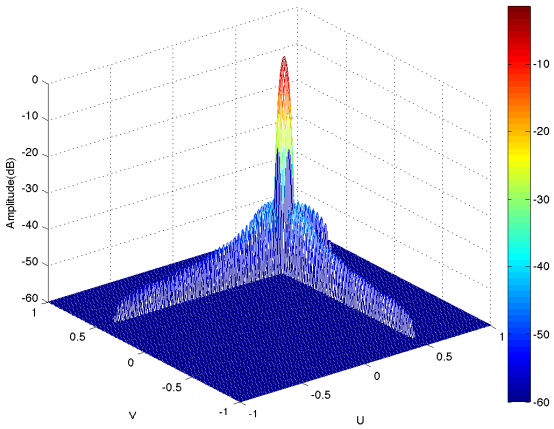


(b) P+: 0.40% optimality gap

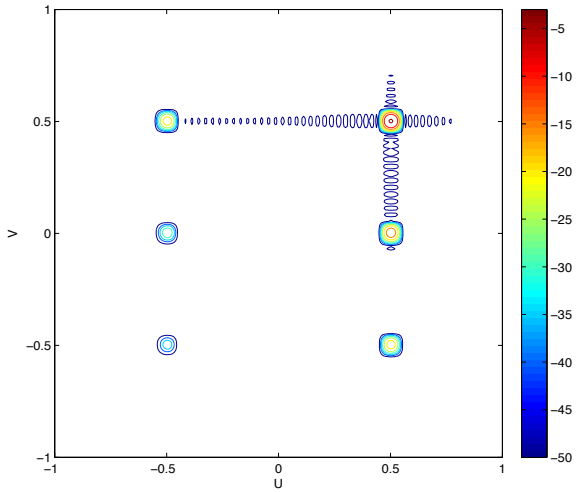
Figure 17: (continued).



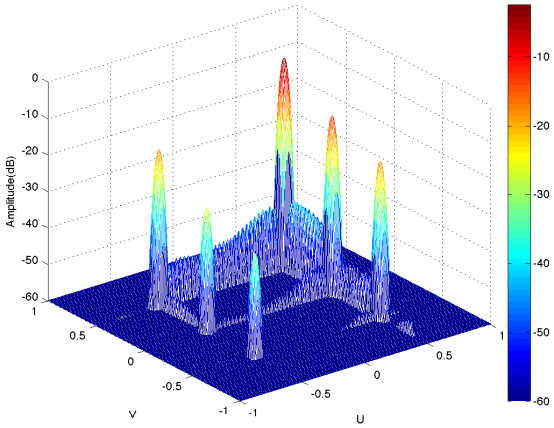
(a) E: contour plot.



(b) E: surface plot

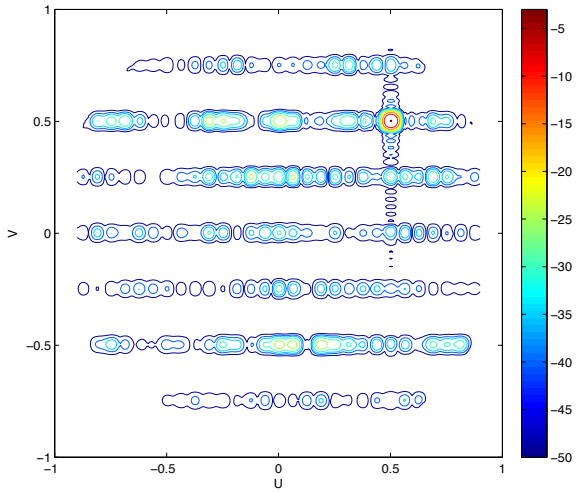


(c) R: contour plot

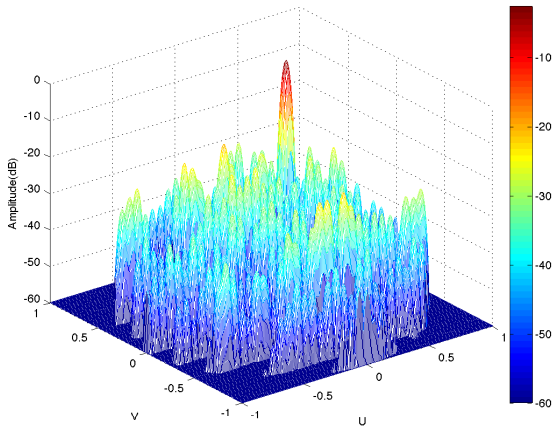


(d) R: surface plot

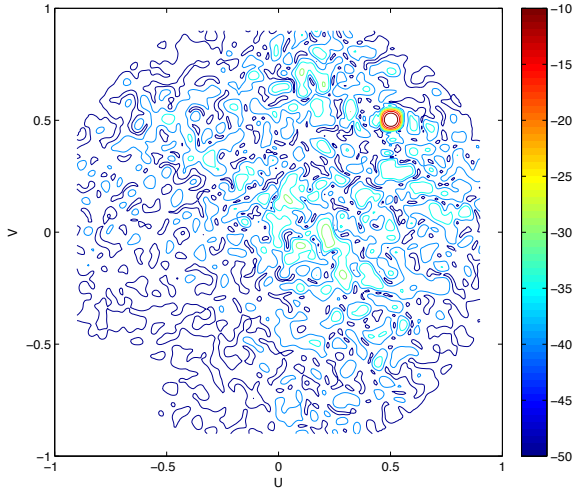
Figure 18: Radiation patterns for 64×64 array size with time delay control at 2×4 rectangular subarray level (**R**), partially optimized octomino subarray level (**P**), and completely optimized octomino subarray level (**P**+).



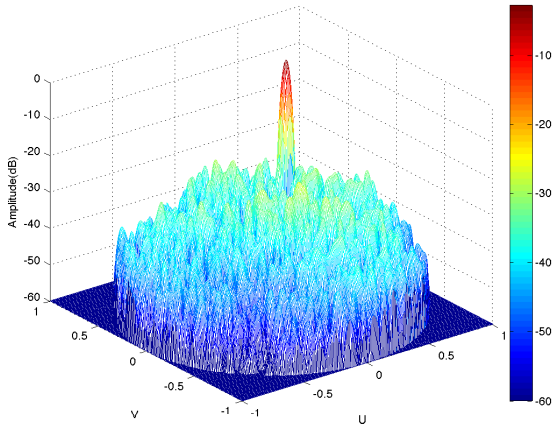
(e) P: contour plot



(f) P: surface plot

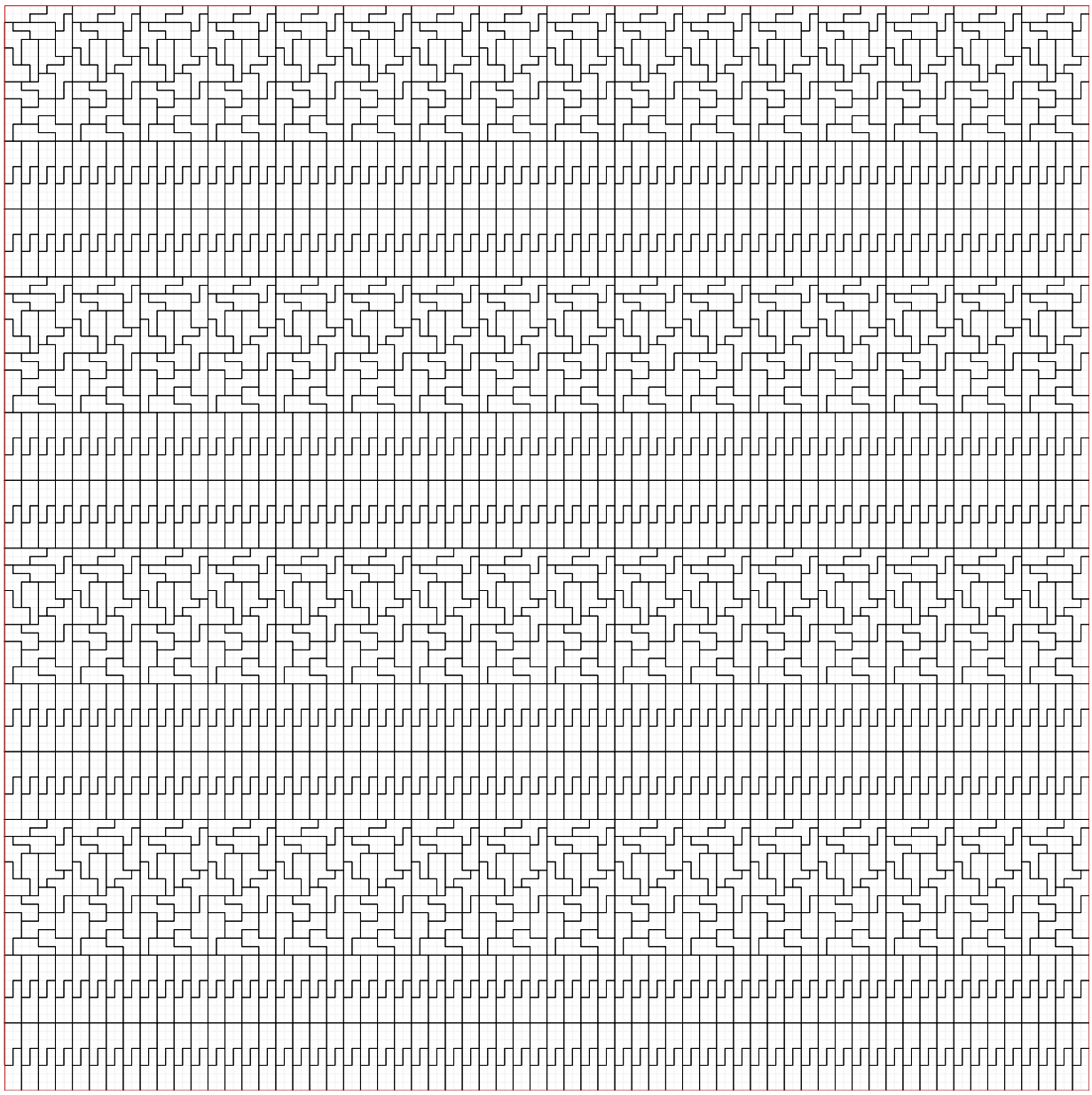


(g) P+: contour plot



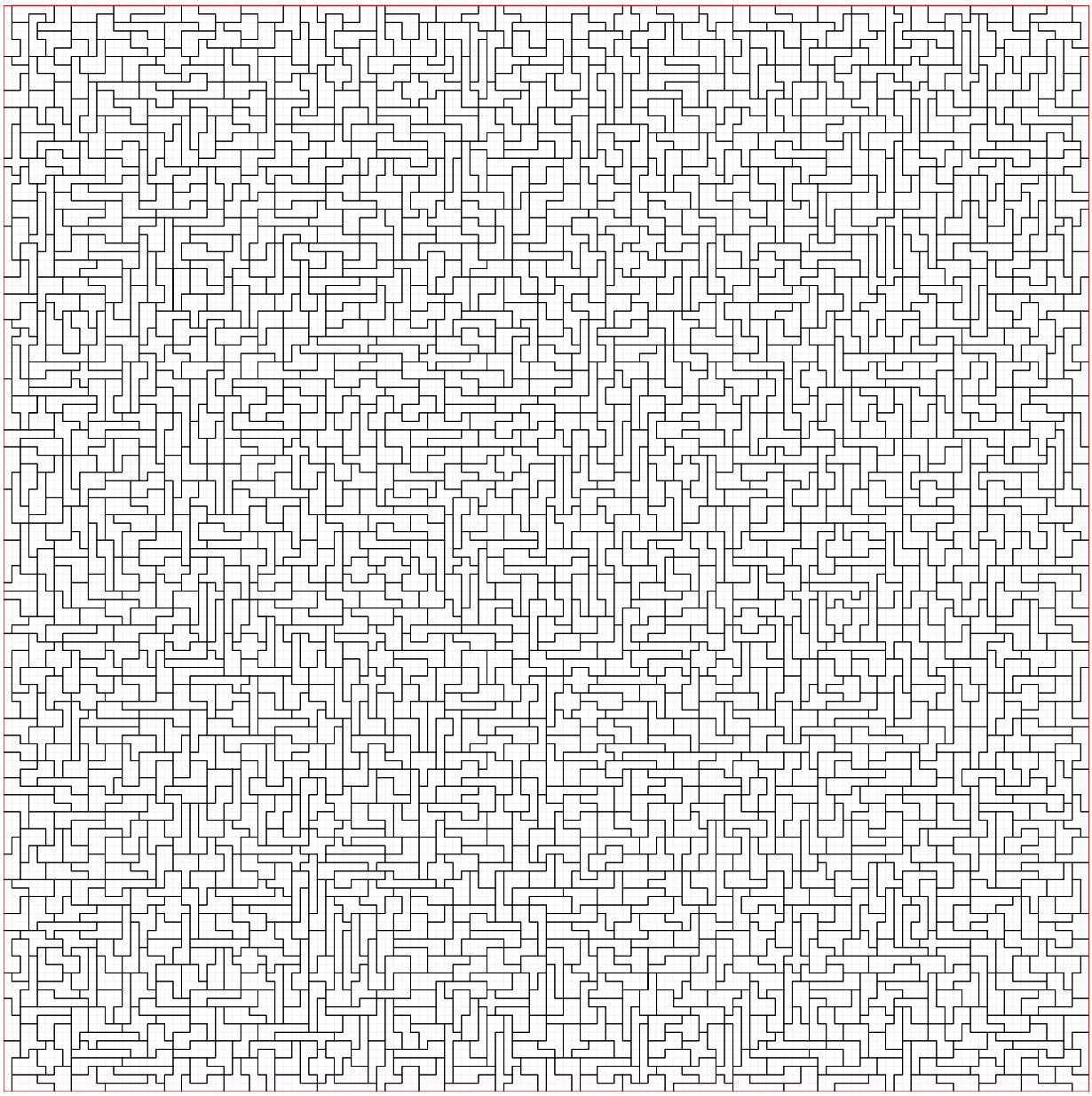
(h) P+: surface plot

Figure 18: (continued).



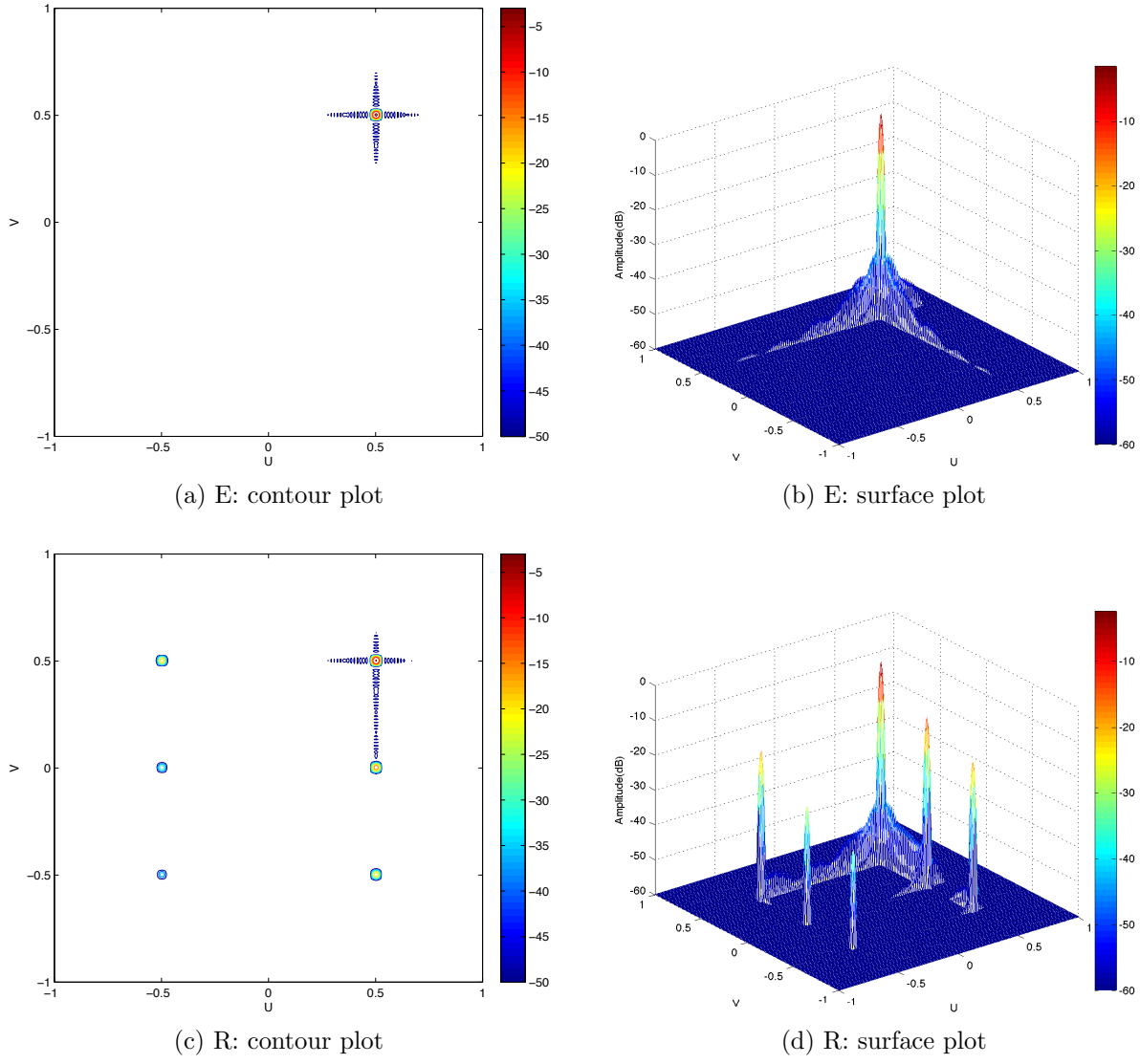
(a) P: 6.32% optimality gap

Figure 19: Partially and completely optimized polyomino tilings for 128×128 array size.



(b) P+: 0.28% optimality gap

Figure 19: (continued).



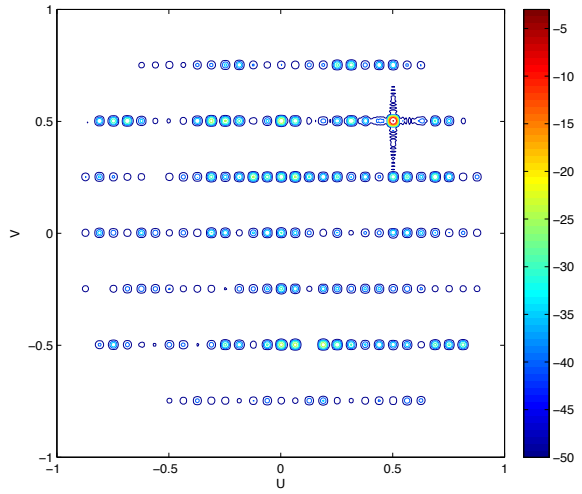
(a) E: contour plot

(b) E: surface plot

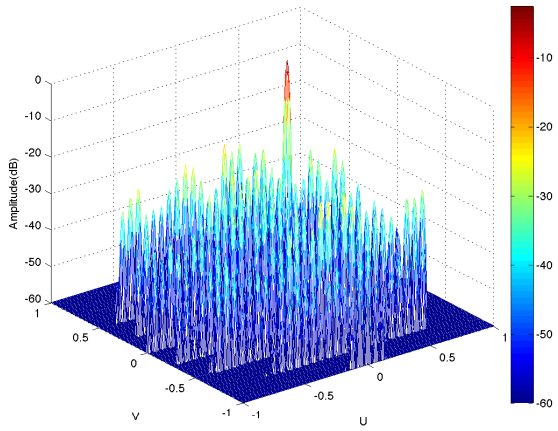
(c) R: contour plot

(d) R: surface plot

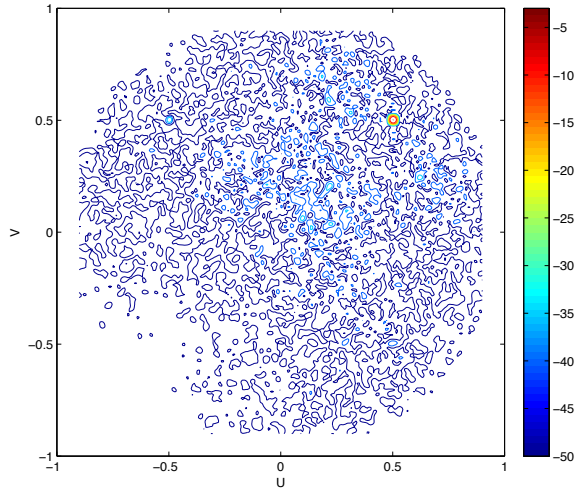
Figure 20: Radiation patterns for 128×128 array size with time delay control at element level (**E**), 2×4 rectangular subarray level (**R**), partially optimized octomino subarray level (**P**), and completely optimized octomino subarray level (**P+**).



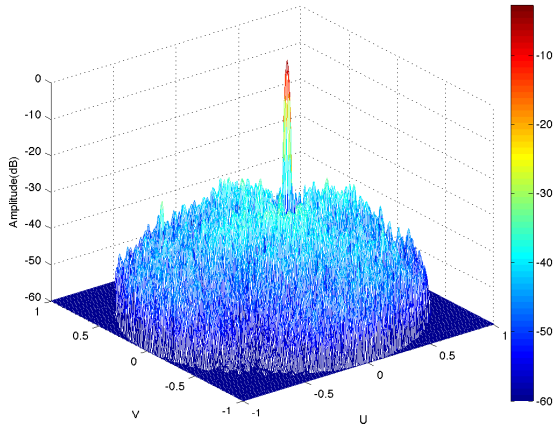
(e) P: contour plot



(f) P: surface plot

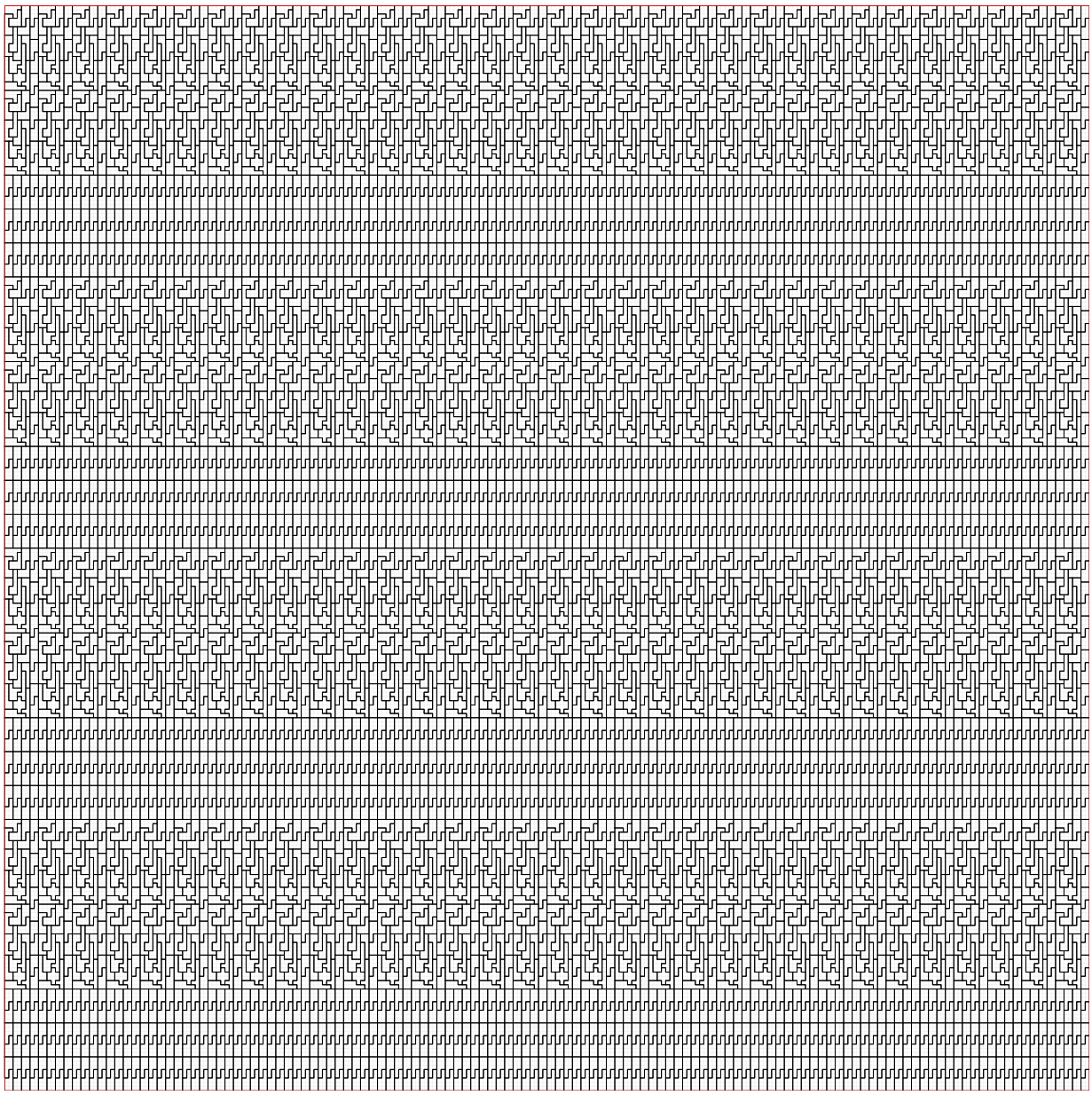


(g) P+: contour plot



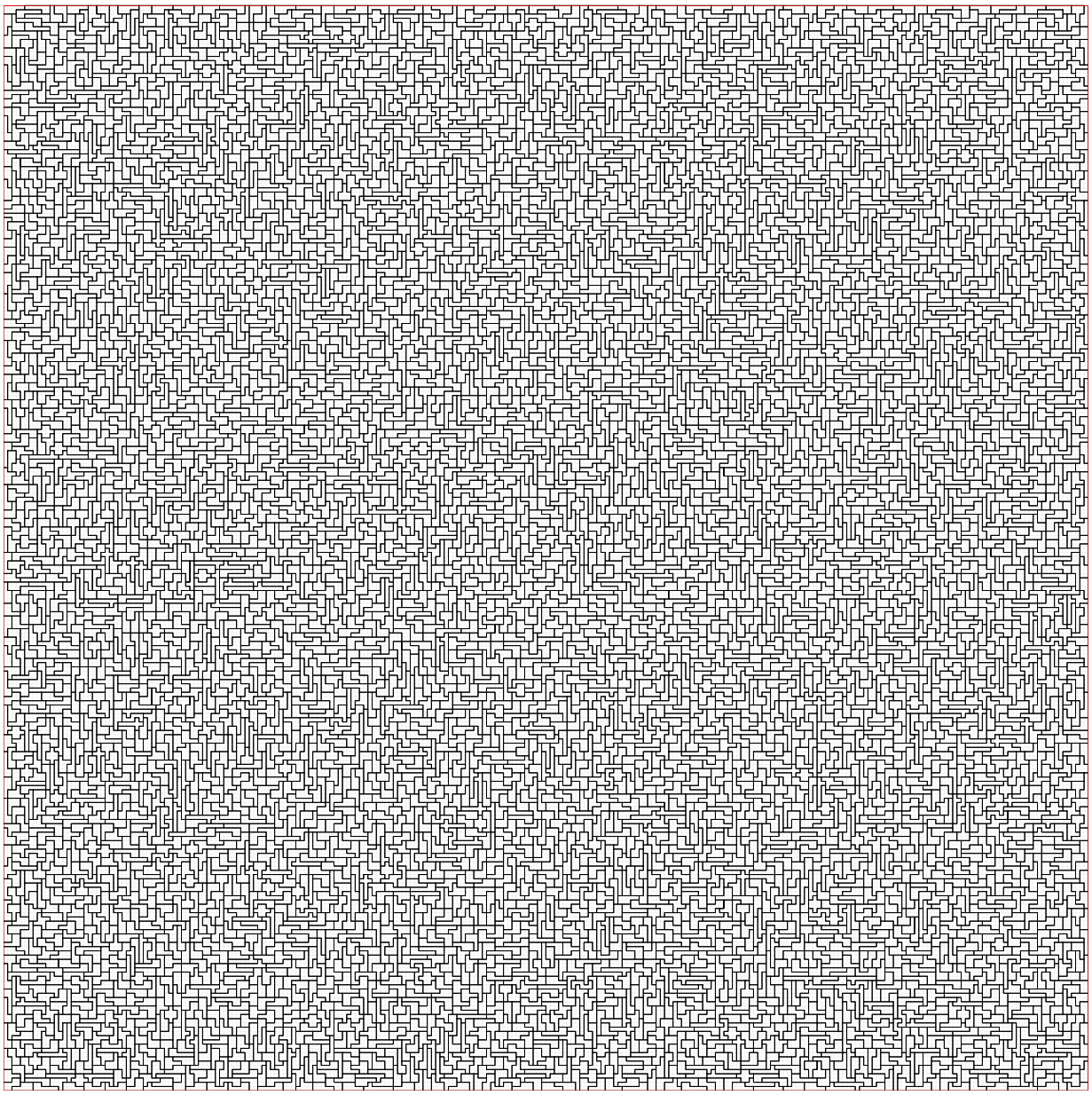
(h) P+: surface plot

Figure 20: (continued).



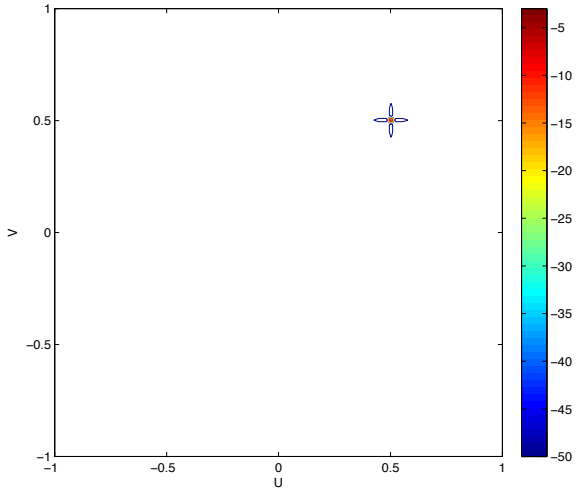
(a) P: 5.97% optimality gap

Figure 21: Partially and completely optimized polyomino tilings for 256×256 array size.

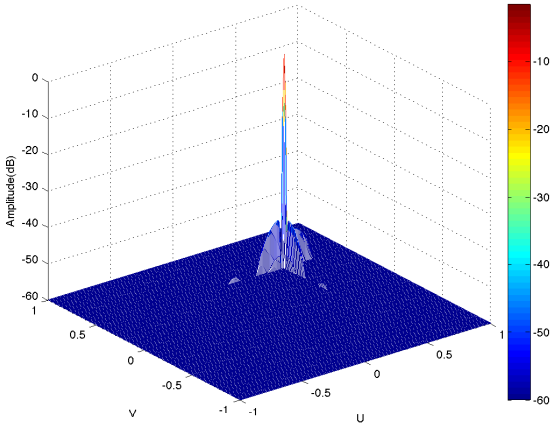


(b) P+: 0.08% optimality gap

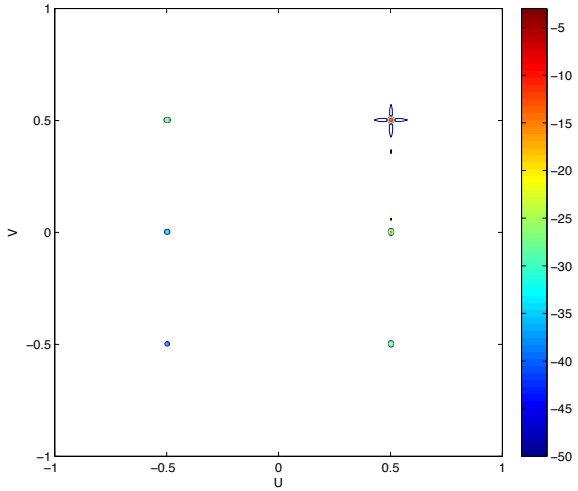
Figure 21: (continued).



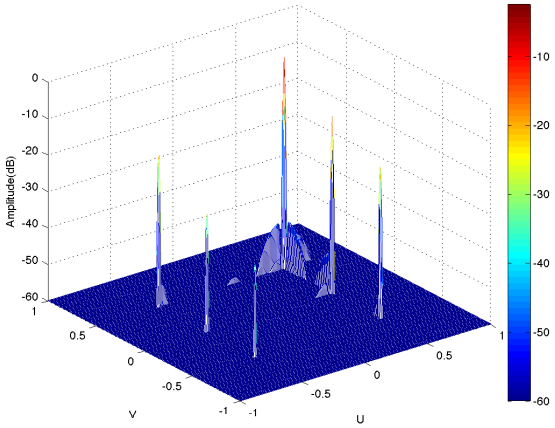
(a) E: contour plot.



(b) E: surface plot

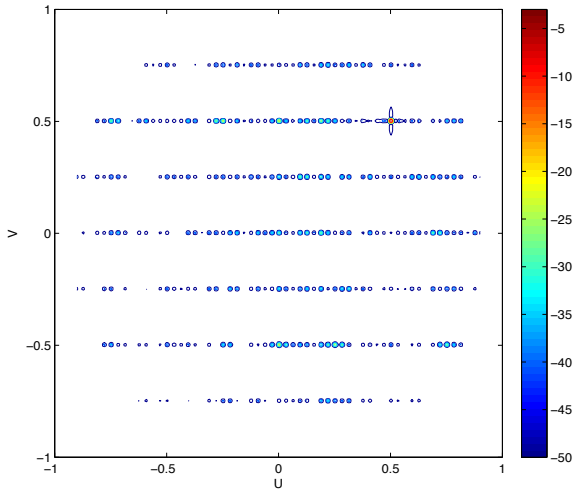


(c) R: contour plot

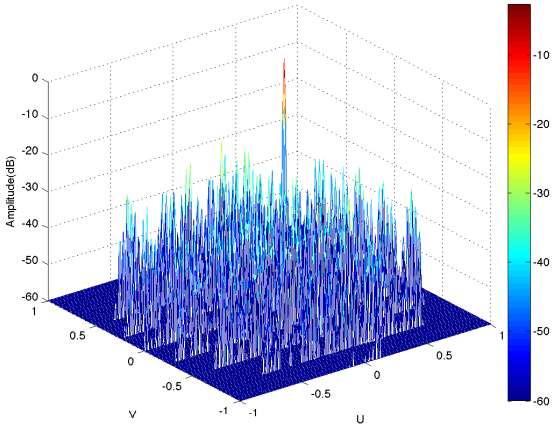


(d) R: surface plot

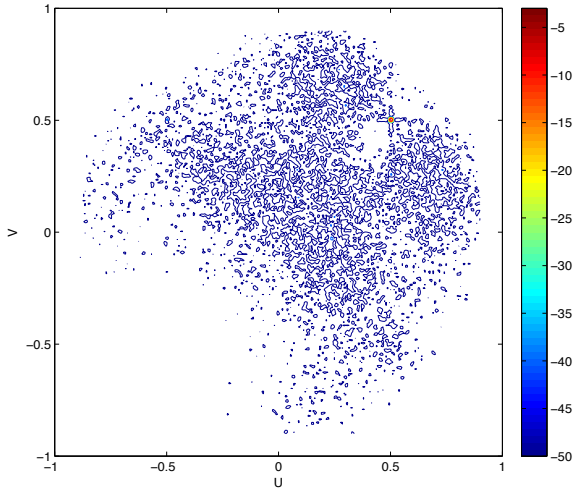
Figure 22: Radiation patterns for 256×256 array size with time delay control at 2×4 rectangular subarray level (**R**), partially optimized octomino subarray level (**P**), and completely optimized octomino subarray level (**P+**).



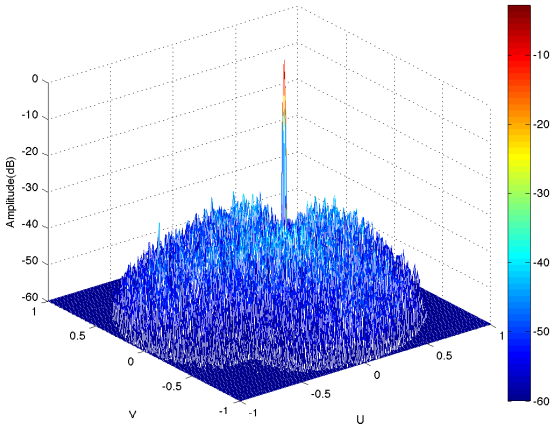
(e) P: contour plot



(f) P: surface plot



(g) P+: contour plot



(h) P+: surface plot

Figure 22: (continued).

2.9 ACKNOWLEDGMENT

We want to thank to Dr. Robert J. Mailloux for his valuable comments on interpretation of the antenna simulation results and to Dr. Scott Santarelli for his assistance with antenna simulation software. We also acknowledge Dr. Arje Nachman and Dr. Donald Hearn for introducing us to the considered application.

3.0 ON GREEDY APPROXIMATION ALGORITHMS FOR A CLASS OF TWO-STAGE STOCHASTIC ASSIGNMENT PROBLEMS

3.1 INTRODUCTION

Given a set V of n agents, a set U of n jobs and a weight (or cost) w_{ij} for each $i \in V$ and $j \in U$, the well-known *linear assignment problem* consists of assigning each agent to exactly one job in such a manner that each job is performed by exactly one of the agents and the total weight (cost) of the obtained assignment is maximized (minimized). In this work we consider the maximization version and the mathematical program for the related linear assignment problem can be given as follows [97]:

$$\max_{x} \quad \sum_{i \in V} \sum_{j \in U} w_{ij} x_{ij} \tag{3.1}$$

$$\text{s.t.} \quad \sum_{j \in U} x_{ij} = 1, \quad \text{for all } i \in V, \tag{3.2}$$

$$\sum_{i \in V} x_{ij} = 1, \quad \text{for all } j \in U, \tag{3.3}$$

$$x_{ij} \in \{0, 1\}, \quad \text{for all } i \in V,\ j \in U. \tag{3.4}$$

Linear assignment problem (3.1)-(3.4) is also known as the *weighted bipartite matching problem*. Namely, given a weighted bipartite graph $G(V \cup U, E)$ with $|V| = |U|$ and arc weights w_{ij} for all $(i, j) \in E$ we need to find a *perfect matching* of maximum weight. Recall that perfect matching is a matching which matches all vertices of the graph.

It is well known that the constraint matrix for (3.2)-(3.3) is totally unimodular [97]. Therefore, we can safely remove integrality constraints (3.4) and solve the linear programming relaxation (3.1)-(3.4) to get the optimal solution. However, the most popular approach to tackle the linear assignment problem is the Hungarian Method [83], which can be considered as an implementation of the primal-dual method for the respective minimum cost flow problem. The Hungarian Method (HM) works with the dual of the linear program (3.1)-(3.3) given by

$$\min_{\alpha,\beta} \quad \sum_{i=1}^{n} \alpha_i + \sum_{j=1}^{n} \beta_j \tag{3.5}$$

$$\text{s.t.} \quad \alpha_i + \beta_j \geq w_{ij}, \quad i = 1, \ldots, n, \quad j = 1, \ldots, n. \tag{3.6}$$

We concider the following two-stage stochastic programming extension of (3.1)-(3.4), which is further referred to as the *two-stage stochastic linear assignment (2SSLA) problem*. Each edge (i, j), $i \in V$ and $j \in U$, is associated with the first-stage weight w_{ij}, and the second-stage weight q_{ij}^k for scenario k, $k = 1, \ldots, K$. The first-stage decision x is to choose some matching in G that is not necessarily perfect. At the second stage a scenario k is realized with probability p_k. For each scenario k, the second-stage decision y^k is to choose a matching over those agents and jobs that are unmatched in the first stage in order to form a perfect matching. The overall goal is to find a perfect matching with the maximum expected weight. Then the two-stage stochastic programming extension of (3.1)-(3.4) can be written as follows:

$$\max_{x,y} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} \cdot x_{ij} + \sum_{k=1}^{K} p_k \cdot \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}^k \cdot y_{ij}^k \tag{3.7}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} \left(x_{ij} + y_{ij}^k\right) = 1, \quad i = 1, \ldots, n, \quad k = 1, \ldots, K, \tag{3.8}$$

$$\sum_{i=1}^{n} \left(x_{ij} + y_{ij}^k\right) = 1, \quad j = 1, \ldots, n, \quad k = 1, \ldots, K, \tag{3.9}$$

$$x_{ij} \in \{0, 1\}, \quad y_{ij}^k \in \{0, 1\}, \quad i, j = 1, \ldots, n, \quad k = 1, \ldots, K. \tag{3.10}$$

2SSLA is partially motivated by the target-based *weapon-target assignment (WTA) problem* [94]. In the WTA problem, a battle manager must assign M weapons to N targets maximizing the expected total damage to the targets. If complete information is initially available, then all weapon-target assignment decisions can be made instantaneously and the resulting problem is referred to as the *static* WTA problem [34, 94]. In the *dynamic* WTA problem, decisions are made over several time periods and the battle manager may not have perfect information about the future. The latter setting naturally leads to a stochastic optimization model.

Over the last decade, there has been a rapid increase in applications of stochastic optimization methodology to a variety of important real-world problems including resource allocation, planning, logistics, scheduling and health care [24, 102]. The key advantage of stochastic programming is that it allows optimal (or near-optimal) decision making with uncertainty considerations, which overcomes many limitations of classical deterministic optimization approaches as it has been demonstrated through a number of studies in the literature [107]. Thus, it is not surprising that algorithmic developments for stochastic integer and combinatorial optimization problems have been among the most active research areas. A number of studies has been focused on the design of exact solutions methodologies, with typical examples in [1, 2, 7, 95, 105, 108].

Unfortunately, in general, stochastic optimization problems are inherently difficult to solve. Many standard deterministic optimization problems that are solvable in polynomial time, become NP-hard if considered in the stochastic environment, e.g., maximum weight matching [81] and spanning tree [52] problems. Moreover, two- and multi-stage stochastic linear programming problems with discrete distributions on the parameters are ♯P-hard and PSPACE-hard, respectively [49]. Thus, we can not expect to be able to solve general stochastic integer and combinatorial optimization problems exactly for large input sizes, and similar to the deterministic optimization literature, it is desirable to design polynomial time approximation algorithms with reasonable performance guarantees. Recent examples of this type of work include methods for solving two-stage stochastic extensions of the shortest path [62], minimum spanning tree [46], min-cut [45, 62], and Steiner tree [66] problems. For more detailed introductions and surveys to stochastic programming, we refer the reader

to [24, 96, 102, 104, 107, 110]. Next we briefly describe two studies that are most closely related to our work and outline the remainder of this chapter.

Kong and Schaefer [81] consider the two-stage stochastic maximum weight matching problem on general graphs. They show that the problem is NP-hard and propose a greedy $\frac{1}{2}$-factor approximation algorithm. Escoffier et al. [50] prove that the two-stage stochastic maximum weight matching problem is APX-complete even for bipartite graphs of maximum degree 4 and general graphs of degree 3, which implies that there is no polynomial-time approximation scheme (PTAS) for this problem as long as P$\neq$ NP. Based on the concepts from [81], they also provide a greedy $\max\{\frac{K}{2K-1}, \frac{\Delta}{2\Delta-1}\}$-approximation algorithm, where K is the number of scenarios in the second-stage and Δ is the degree of the bipartite graph.

Our work is essentially built on these two studies [50, 81]. In Section 3.2, we consider the greedy approximation methods from their work for the two-stage stochastic linear assignment problem. Since the maximum weight matching problem on bipartite graphs can be easily reduced to the linear assignment problem via addition of dummy agents and/or jobs, the 2SSLA problem is also APX-complete. Section 3.3 covers a necessary optimality condition that generalizes and unifies the key ideas behind the two algorithms by Kong and Schaefer [81] and Escoffier et al. [50]. Then based on this optimality condition, in Section 3.4, we design a new greedy approximation algorithm referred to as EGA. While the developed approach preserves the existing approximation guarantees, we are not able to prove whether EGA provides a strictly better approximation bound. However, analytical observations in Section 3.4.4 and extensive computational results in Section 3.5 indicate that EGA has strictly better results on some rather broad classes of the two-stage stochastic linear assignment problems.

3.2 GREEDY APPROXIMATION ALGORITHMS

3.2.1 Basic Greedy Approach

First, we discuss Greedy Algorithm (GA) for the more general two-stage stochastic matching problem given in [81]. Because the linear assignment is a specific case of the maximum weight matching problem, this algorithm can be simply adopted to 2SSLA. We need the following notation:

Definition 1. *A first-stage myopic solution is an optimal solution to:*

$$(\mathbf{GA-I}) : \max \left\{ \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_{ij} \;\middle|\; \forall j \;\; \sum_{i=1}^{n} x_{ij} = 1, \;\; \forall i \;\; \sum_{j=1}^{n} x_{ij} = 1; \;\; \forall i,j \;\; x_{ij} \in \{0,1\} \right\}. \quad (3.11)$$

Definition 2. *A second-stage myopic solution for scenario k is an optimal solution to:*

$$(\mathbf{GA-II}) : \max \left\{ \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij}^{k} y_{ij}^{k} \;\middle|\; \forall j \;\; \sum_{i=1}^{n} y_{ij}^{k} = 1, \;\; \forall i \;\; \sum_{j=1}^{n} y_{ij}^{k} = 1; \;\; \forall i,j \;\; y_{ij}^{k} \in \{0,1\} \right\}. \quad (3.12)$$

First- and second-stage myopic solutions correspond to deterministic linear assignment problems with the appropriate choices of weights in the objective functions. Let $\mathbf{x}^{GA}$ and Z_1^{GA} be a first-stage myopic solution and the respective optimal objective function value. Similarly, let $\mathbf{y}_k^{GA}$ and Z_{2k}^{GA} be a second-stage myopic solution and the respective optimal objective function value for scenario k. Finally, denote by Z_2^{GA} the expected value of the second-stage myopic solutions, i.e.,

$$Z_2^{GA} = \sum_{k=1}^{K} p_k Z_{2k}^{GA}.$$

Initially, GA finds a first-stage myopic solution (GA-I) and second-stage myopic solutions for each scenario (GA-II). Then it compares the objective function value of the first-stage myopic solution with the expected objective function value of the second-stage myopic solutions. The final assignment weight Z^{GA} corresponds to the better of them, i.e.,

$$Z^{GA} = \max \left\{ Z_1^{GA}, \;\; Z_2^{GA} \right\}. \quad (3.13)$$

Thus, the final assignments are given either by $(\mathbf{x}^{GA}, \mathbf{0}, \ldots, \mathbf{0})$, or by $(\mathbf{0}, \mathbf{y}_1^{GA}, \ldots, \mathbf{y}_K^{GA})$. In other words, all assignments are made completely either at the first stage, or at the second stage for each scenario.

Theorem 1 ([81]). *Greedy Algorithm is an approximation algorithm with the performance guarantee* $\frac{1}{2}$ *for the 2SSLA problem.*

Observe that GA solves the deterministic linear assignment problem $K+1$ times, e.g., using the Hungarian Method (HM) [83]. Consequently, given the complexity of HM, the outlined algorithm obtains $\frac{1}{2}$–approximate solution of (3.7)-(3.10) in $O(Kn^3)$ arithmetic operations.

3.2.2 Greedy Approach of Escoffier et al. [50]

A slightly more advanced greedy approach, further referred to as GAE, is proposed by Escoffier et al. in [50]. The basic idea is that if $\sum_{k=1}^{K} p_k q_{ij}^k > w_{ij}$ for some i and j, then it can not be optimal to assign agent i to job j in the first stage. This result follows from the observation that any solution to the 2SSLA problem that assigns i to j in the first stage, could be improved by assigning i to j in the second stage across all scenarios. In fact, as it is demonstrated in Section 3.3, this result is a special case of a more general necessary optimality condition.

Initially, GAE replaces all first-stage weights w_{ij} with

$$\hat{w}_{ij} = \max \left\{ w_{ij}, \ \sum_{k=1}^{K} p_k q_{ij}^k \right\}$$

and obtains the first-stage myopic solution with the updated weights. Then, all agent-job assignments (i, j), i.e., $j = mate[i]$ and $i = mate[j]$, such that $\hat{w}_{ij} = \sum_{k=1}^{K} p_k q_{ij}^k$ are "moved" to the second stage. Subsequently, GAE solves K assignment problems for all agents and jobs moved to the second stage across all scenarios. Denote the resulting solution by Z_1^{GAE} and the above described algorithmic procedure by GAE-I. Next GAE compares Z_1^{GAE} with the expected objective function value of the second-stage myopic solutions Z_2^{GA}. The final assignment weight Z^{GAE} corresponds to the better of them, i.e.,

$$Z^{GAE} = \max \left\{ Z_1^{GAE}, \ Z_2^{GA} \right\}.$$

Theorem 2 ([50]). *GAE is an approximation algorithm with the performance guarantee* $\frac{K}{2K-1}$ *for 2SSLA.*

Theorem 3 ([50]). *GAE is an approximation algorithm with the performance guarantee* $\frac{\Delta}{2\Delta-1}$ *for 2SSLA, where* Δ *is the degree of the bipartite graph.*

Both of the above approximation bounds are slightly better than $\frac{1}{2}$ approximation bound of GA. The running time of GAE is $O(Kn^3)$.

3.3 NECESSARY OPTIMALITY CONDITION

Let $A \subseteq V$ be a subset of agents and $J \subseteq U$ be a subset of jobs such that $|A| = |J|$. Consider a two-stage stochastic linear assignment problem on these subsets of agents and jobs. Let $W_1[A, J]$ be the first-stage myopic solution and $W_2[A, J]$ be the expected value of the second-stage myopic solutions over all scenarios.

Proposition 11 (Necessary Optimality Condition). *Let* $A \subseteq V$, $J \subseteq U$ *and* $|A| = |J|$. *If* $W_1[A, J] < W_2[A, J]$ *(*$W_1[A, J] > W_2[A, J]$*), then no optimal solution of 2SSLA can contain a perfect assignment between agents in* A *and jobs in* J *in the first stage (second stage).*

Proof. Consider an optimal solution of an instance of the 2SSLA problem. If $W_1[A, J] < W_2[A, J]$ ($W_1[A, J] > W_2[A, J]$) and the optimal solution contains a perfect assignment between A and J in the first stage (second stage), then moving assignments between A and J to the second stage (first stage) increases the weight of the current solution, which contradicts our assumption about its optimality. □

Unfortunately, the optimality condition of Proposition 11 is not sufficient for optimality even if it is checked for all $O(2^n)$ possible subsets of V and U. Consider a simple instance of the 2SSLA problem given in Figure 23. In the first stage, agents a_1 and a_2 are assigned to jobs j_2 and j_1, respectively. In the second stage, agent a_3 is assigned to job j_3 under both scenarios. The total weight of the assignment is 4 units. Notice that this solution does not violate the necessary optimality condition for any subset of agents and jobs. However, optimal assignment has a total weight of 5 units which is achieved by assigning a_1 to j_1, a_2 to j_3, and a_3 to j_2 in the first stage.

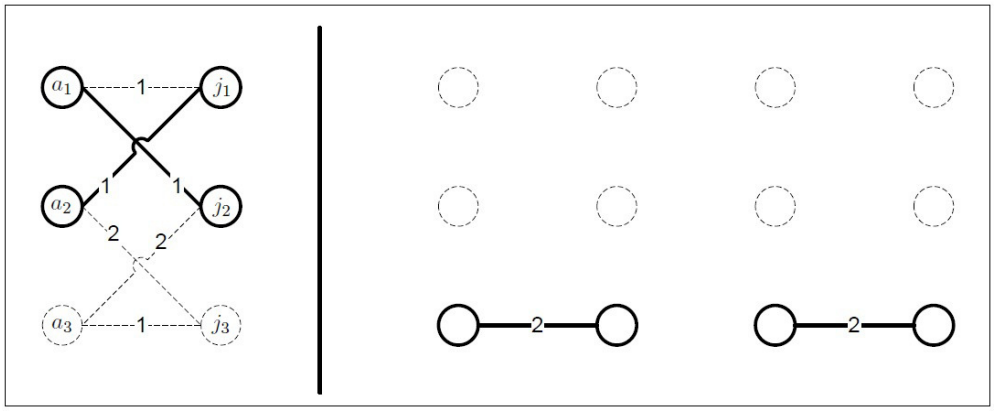


Figure 23: A counterexample to show that the necessary optimality condition given by Proposition 11 is not sufficient for optimality. Only arcs with nonzero weight are shown.

Nevertheless, Proposition 11 can be applied to construct approximation algorithms. In fact, algorithms GA (Section 3.2.1) and GAE (Section 3.2.2) are based on the necessary optimality condition for some specific subsets of agents and jobs. Observe that GA is the implementation of Proposition 11 when $|A| = |J| = n$. In other words, GA verifies the necessary optimality condition only for $A = V$ and $J = U$. If $W_1[V, U]$ (i.e., the weight of the first-stage myopic solution given by GA-I) is greater than $W_2[V, U]$ (i.e., the expected weight of the second-stage myopic solutions over all scenarios given by GA-II), then it moves assignments between V and U to the second stage. Otherwise, all assignments are made in the first stage.

Similarly, GAE is the implementation of the necessary optimality condition for all sets A and J such that $|A| = |J| = 1$ and $|A| = |J| = n$. Recall that GAE-I moves assignment (i, j) to the second stage if it has a better expected weight in the second stage. Thus, $A = \{i\}$, $J = \{j\}$, $W_1[A, J] = w_{ij}$, and $W_2[A, J] = \sum_{k=1}^{K} p_k q_{ij}^k$. Next, GAE compares Z_1^{GAE} with $Z_2^{GA} = W_2[V, U]$ and outputs the better solution, which is equivalent to checking the necessary optimality condition for $|A| = |J| = n$. The only difference is that instead of $W_1[V, U]$, we use the solution of GAE-I and compare it to $W_2[V, U]$.

3.4 ENHANCED GREEDY APPROACH

In this section we propose a more generic approximation approach, further referred to as Enhanced Greedy Algorithm (EGA). The key idea is to apply the necessary optimality condition described by Proposition 11 in a somewhat more sophisticated manner. Also, EGA uses the Hungarian Method as a standard routine to solve all the deterministic linear assignment subproblems.

Consider the dual problem of the LP relaxation of (3.7)-(3.9) given by:

$$\min_{\alpha,\beta} \quad \sum_{i=1}^{n}\sum_{k=1}^{K}\alpha_{ik} + \sum_{j=1}^{n}\sum_{k=1}^{K}\beta_{jk} \tag{3.14}$$

$$\text{s.t.} \quad \sum_{k=1}^{n}(\alpha_{ik}+\beta_{jk}) \geq w_{ij}, \qquad i=1,\ldots,n, \quad j=1,\ldots,n, \tag{3.15}$$

$$\alpha_{ik}+\beta_{jk} \geq p_k q_{ij}^k, \qquad i=1,\ldots,n, \quad j=1,\ldots,n, \quad k=1,\ldots,K. \tag{3.16}$$

In the remainder of this chapter we refer to (3.15) and (3.16) as the "first-stage" and "second-stage" dual constraints, respectively. Furthermore, for convenience of notation, let $\tilde{q}_{ij}^k = p_k q_{ij}^k$.

EGA has two major steps. The first step (referred to as EGA-I) is to start with a first-stage myopic solution and then attempt to improve the objective function value by "moving" some of the assignments to the second stage via checking the necessary optimality condition. The second step of the EGA (referred to as EGA-II) is to start with second-stage myopic solutions and then attempt to improve the objective function value by "moving" some of the assignments to the first stage. Subsequently, EGA chooses the solution with the better objective function value and outputs it as the final solution.

Note that there is some relationship between the necessary optimality condition given by Proposition 11 and the feasibility of the dual program (3.15)-(3.16). Any first-stage myopic solution is feasible to the first-stage dual constraints (3.15) and second-stage myopic solutions are feasible to the second-stage dual constraints (3.16). However, the myopic solutions are not necessarily feasible to both (3.15) and (3.16) simultaneously. Thus, EGA-I starts with the

first-stage myopic solution feasible to the first-stage dual constraints, and uses the necessary optimality condition to achieve feasibility of the second-stage dual constraints. Specifically, it will be shown that the necessary optimality condition for specific pairs of subsets of agents and jobs corresponds to some second-stage aggregated dual constraint that is obtained by aggregating the respective subset of the second-stage dual constraints. Similarly, EGA-II starts with the second-stage myopic solutions feasible to the second-stage dual constraints, and uses the necessary optimality condition to achieve feasibility of some first-stage aggregated dual constraint that is obtained by aggregating a subset of the first-stage dual constraints.

3.4.1 Improving the first-stage assignment (EGA-I)

In this section we describe the key ideas behind EGA-I. The pseudo-code of the approach is given by Algorithm 4. The initial step of the method follows GAE (Section 3.2.2). Specifically, EGA-I applies the necessary optimality condition (Proposition 11) and considers sets of unit cardinality, i.e., $|A| = |J| = 1$. For every agent-job pair (i, j), $A = \{i\}$ and $J = \{j\}$, let $W_1[\{i\}, \{j\}] = w_{ij}$ and $W_2[\{i\}, \{j\}] = \sum_k \tilde{q}_{ij}^k$. Then, the algorithm performs the following weight update in the first stage:

$$\hat{w}_{ij} = \max\left\{W_1[\{i\}, \{j\}],\, W_2[\{i\}, \{j\}]\right\} = \max\left\{w_{ij},\, \sum_k \tilde{q}_{ij}^k\right\}. \tag{3.17}$$

Subsequently, HM is applied to the resulting graph, all assignments with a modified weight are moved to the second stage, and all edge weights are reset to their original values (lines 1 and 2 of Algorithm 4). We want to emphasize that after this point, EGA-I considers only agents and jobs that are not moved to the second stage in line 1.

Next EGA-I checks the optimality condition for sets with cardinality greater than 1, i.e., $|A| = |J| > 1$. Observe that there are $O(2^n)$ possible combinations of A and J. Furthermore, in order to compute $W_1[A, J]$ and $W_2[A, J]$, one needs to solve $K + 1$ deterministic linear assignment problems for every subset. Because performing all of the above steps is computationally prohibitive, EGA-I considers only a few subsets that are promising and easy to check.

Algorithm 4: `EGA-I`

Input: n agents, n jobs, K scenarios, w_{ij}, q_{ij}^k, p_k

Run GAE-I
Reset all the first-stage weights to their original values and remove from consideration all agents and jobs moved to the second stage by GAE-I
Let $\tilde{q}_{ij}^k = p_k q_{ij}^k$; define G_0 and G_k to be the graphs for the first stage and the k^{th} scenario in the second stage, respectively
Run `Hungarian Method` on G_k for all k
Let C include closed subsets of agents and jobs in the obtained second-stage solution
foreach $\{A, J\} \in C$ **do**
    Let $\Delta = W_2[A, J] - (\sum_{i \in A} \alpha_i + \sum_{j \in J} \beta_j)$, where α, β is the dual solution for G_0
    if $\Delta > 0$ **then**
        Let G be the graph containing only A and J
        Run `Hungarian Method` on G
        Let $\tilde{E}$ be set of edges selected in the resulting assignment
        Let $\tilde{E} = \{e_1, e_2, \dots e_{|A|}\}$ be sorted in non-decreasing order of edge weight
        Set $w_{e_{|A|+1}} = \infty$ and let $w \in \mathbb{R}_+$ and $t \in \mathbb{Z}_+$ satisfy the following condition:
        (1) $t \cdot w - \sum_{r=1}^{t} w_{e_r} = \Delta$
        (2) $w_{e_t} \leq w < w_{e_{t+1}}$
        Set $w_{e_r} = w$ for $r = \overline{1, t}$ in G_0
    else
        $C \leftarrow C \backslash \{A, J\}$
Run `Hungarian Method` on G_0.
begin Reset G_0
    while *there exists* $\{A, J\} \in C$ *not closed in* G_0 **do**
        foreach *Edge weight* w_{ij} *modified in* $\{A, J\}$ **do**
            Reset w
            Perform one iteration of `Hungarian Method`
        $C \leftarrow C \backslash \{A, J\}$
Move all closed subsets in C to the second stage
Redefine G_k to be the subgraph of agents and jobs moved to the second-stage for scenario k.
Run `Hungarian Method` on G_k for all k
$Z_1^{EGA} = \sum_{i \in G_0} w_{i,mate[i]} + \sum_k \sum_{i \in G_k} \tilde{q}_{i,mate_k[i]}^k$
return Z_1^{EGA}, G_0, *and* $G_k \ \forall \ k$

Definition 3 (Closed Subset)**.** *A closed subset in the first stage is a pair of subsets A of agents and J of jobs such that all agent-job assignments remain within these two sets in the first-stage myopic solution, i.e.,*

$$J = \{j \in U \mid j = mate[i] \text{ for some } i \in A\},$$
$$A = \{i \in V \mid i = mate[j] \text{ for some } j \in J\}.$$

A closed subset in the second stage is a pair of subsets A of agents and J of jobs such that all agent-job assignments remain within these two sets across all scenarios in the second-stage myopic solutions, i.e.,

$$J = \{j \in U \mid j = mate_k[i] \text{ for some } i \in A \text{ and some scenario } k\},$$
$$A = \{i \in V \mid i = mate_k[j] \text{ for some } j \in J \text{ and some scenario } k\}.$$

Here, we want to provide some details about how EGA-I finds closed subsets. For the first-stage myopic solution, given a subset of agents A, we simply construct J from the mates of agents in A. For the second stage, EGA-I starts constructing a closed subset in the second stage with empty sets A of agents and J of jobs. Given the second-stage myopic solution, we arbitrarily choose an agent and add it to the set of agents A. Then, all jobs that this agent is assigned to across various scenarios are added to the set of jobs J. Notice that an agent may be assigned to the same job in several scenarios. Then, for all jobs that are selected in the previous step, the algorithm updates A finding all agents that jobs from J are assigned to across all scenarios. The algorithm continues in this manner until both sets cease to change, which implies that a closed subset is constructed. The whole process described above is repeated for the remaining agents and jobs until a partition of the set of agents and jobs into a set of closed subsets is obtained.

If A and J correspond to one of the closed subsets in the second stage found by our algorithm, then $|A| = |J|$ by the definition of a closed subset. Moreover, the value of $W_2[A, J]$ is readily available from the second-stage myopic solutions. After finding closed subsets in the second stage, EGA-I identifies the ones that satisfy (line 7):

$$\sum_{i \in A} \alpha_i + \sum_{j \in J} \beta_j < W_2[A, J], \tag{3.18}$$

where α_i and β_j are dual variables associated with the first-stage myopic solution. Clearly, closed subsets that satisfy (3.18) violate the necessary optimality condition since the left-hand side of (3.18) is an upper bound on $W_1[A, J]$. We use the dual solution due to the fact that the pair (A, J) is not necessarily closed in the first stage and we do not want to solve an assignment problem to find $W_1[A, J]$.

For all subsets that are closed in the second stage and satisfy (3.18), EGA-I updates the first-stage weights (lines 6-18). In contrast to GAE-I, updating these weights properly turns out to be a more difficult task when we have more than one agent-job pair to consider. Our main goal is to update edge weights in the first stage for each closed subset of the second stage in such a way that: (1) the resulting assignment with updated weights should favor the sets to be also closed in the first stage and (2) if the set becomes also closed in the first stage, then the weight of the assignment within this set should be exactly $W_2[A, J]$.

Next we discuss in detail the weight update procedure for pair (A, J). First, $W_1[A, J]$ is computed by running HM. Define $\tilde{E}$ to be the set of all agent-job assignments in the obtained solution. Then we increase the weights w_{ij} in the first stage only for pairs $(i, j) \in \tilde{E}$ according to the following procedure (lines 10-16).

- Let $\tilde{E} = \{e_1, \ldots, e_{|A|}\}$ be sorted in non-decreasing order of the edge weights.
- Find $w \in \mathbb{R}_+$ and $t \in \mathbb{Z}_+$, $t \leq |A|$, that satisfy the following conditions:

$$\sum_{r=1}^{t}(w - w_{e_r}) \;=\; t \cdot w - \sum_{r=1}^{t} w_{e_r} \;=\; \Delta, \tag{3.19}$$

$$w_{e_t} \;\leq w \;<\; w_{e_{t+1}}, \tag{3.20}$$

  where we assume that $w_{e_{|A|+1}} = +\infty$.
- Set the weight of each $e_r \in \tilde{E}$, $1 \leq r \leq t$, to be w.

Let $\tilde{W}_1[A, J]$ be the weight of the optimal assignment in (A, J) after the weight update procedure described above. It is easy to observe that $\tilde{W}_1[A, J] = W_1[A, J] + \Delta = W_2[A, J]$. Simply speaking, we increase the weights of the edges with the smallest weights until we have the total increase of Δ.

As an example, assume that $\{3, 10, 25\}$ are the weights of the assignment in $\tilde{E}$ (i.e., we have 3 agents and 3 jobs and the selected assignment edges have weights 3, 10, and

25). Thus, $W_1[A, J] = 38$. Let $W_2[A, J] = 50$. Then, $\Delta = 12$. If we start with $t = 1$, then by (3.19), we get $w = 15$, which violates (3.20). Incrementing t, we set $t = 2$ and find $w = 12.5$, which satisfies (3.20). Thus, we set $w_{e_1} = w_{e_2} = 12.5$. Now we have $\tilde{W}_1[A, J] = 12.5 + 12.5 + 25 = 50 = W_2[A, J]$.

Subsequently, EGA-I finds the first-stage myopic solution with the updated weights (line 19). For every closed subset (A, J), we check whether it remains closed in the first-stage, i.e., $mate[i] \in J \ \forall i \in A$. If this is not the case, we restore each modified assignment weight of $[A, J]$ and perform one iteration of HM to restore optimality. This phase ends when all the remaining closed subsets are also closed in the first stage. Note that the number of weights restored and the number of HM iterations performed are at most n. Next, the remaining closed sets are moved to the second stage (line 26). Finally, EGA-I solves K deterministic assignment problems with all the agents and jobs moved to the second stage (including those moved after line 1) to find the second stage solution, and outputs the resulting assignment.

Lemma 1. *Let Z_1^{EGA} be the weight of the assignment returned by EGA-I. Then*

$$Z_1^{EGA} \geq Z_1^{GAE} \geq Z_1^{GA} \ . \tag{3.21}$$

Proof. It is clear that the solution found after line 1 is exactly the solution found by GAE-I. Next consider the first-stage solution and edge weights after line 25. Let $\{A, J\} \in C$ be a closed subset that is not removed from consideration during the procedure between lines 20-25. Any assignment edge in the first stage that does not belong to closed subsets from C, now has its original value as it was reset in line 23. Any assignment edge weight that belongs to a closed subset $\{A, J\} \in C$ is at least as large as its original value because the weight update mechanism given by (3.19)-(3.20) only increases the weights of the edges. By construction, since $\{A, J\} \in C$ is also closed in the first stage, the total weight of the assignments in this subset, $\tilde{W}_1[A, J] (> W_1[A, J])$, is exactly equal to $W_2[A, J]$. It implies that there exists an assignment of A and J at the second stage with the same weight. Hence, the weight of the assignment returned after line 25 is at least as large as weight of the assignment returned after line 1. The necessary result follows. □

3.4.2 Improving the second-stage assignment (EGA-II)

EGA-II starts with all agent-job assignments made in the second stage and then attempts moving some of them to the first stage if it is worth doing so. Due to the lack of control in preserving closed subsets after each weight update in the second stage, this approach is more sensitive to variations in assignments across the scenarios. Using the necessary optimality condition, EGA-II tries to achieve dual feasibility of the model given by (3.15) and (3.16). First, it solves the optimization problem given by the objective function (3.14) and the constraint set (3.16). Since the constraints are separable for scenarios, we use HM to solve the assignment problem for each scenario separately (line 3 of Algorithm 5). Therefore, initially all second-stage constraints of the form

$$\alpha_{ik} + \beta_{jk} \geq \tilde{q}^k_{ij}, \qquad \forall i, j, k \tag{3.22}$$

are satisfied as the obtained assignments are the second-stage myopic solutions. Then the algorithm searches for a pair (i', j') such that

$$(i', j') = \arg\max_{(i,j)} \left\{ w_{ij} - \sum_{k=1}^{K} (\alpha_{ik} + \beta_{jk}) \;\middle|\; w_{ij} - \sum_{k=1}^{K} (\alpha_{ik} + \beta_{jk}) > 0 \right\}.$$

If such pair (i', j') does not exists, then the algorithm stops. Otherwise, it implies that the first-stage dual constraint $\left\{ \sum_{k=1}^{K} (\alpha_{i'k} + \beta_{j'k}) \geq w_{i'j'} \right\}$ is violated. Then consider slack $s_k = \alpha_{i'k} + \beta_{j'k} - \tilde{q}^k_{i'j'}$ for each scenario (line 8). EGA-II updates the weight $\tilde{q}^k_{i'j'}$ according to the following scheme:

- If $\sum_k s_k > 0$:

$$\hat{q}^k_{i'j'} = \tilde{q}^k_{i'j'} + \left(w_{i'j'} - \sum_k \tilde{q}^k_{i'j'} \right) \left(\frac{s_k}{\sum_k s_k} \right) . \qquad \text{(line 11)}$$

- If $\sum_k s_k = 0$:

$$\hat{q}^k_{i'j'} = w_{i'j'} \left(\frac{\tilde{q}^k_{i'j'}}{\sum_k \tilde{q}^k_{i'j'}} \right) . \qquad \text{(line 14)}$$

The intuition behind these update strategies is attempting to keep agent i' assigned to job j' across all scenarios after we update arc weights $q^k_{i'j'}$. Note that after the weight update, we have

$$\sum_{k=1}^{K} \hat{q}^k_{i'j'} \;=\; w_{i'j'}.$$

Therefore, any labeling feasible for the second-stage dual constraints $\left\{\; \alpha_{i'k} + \beta_{j'k} \geq \hat{q}^k_{i'j'} \;\forall k \;\right\}$ is also feasible for the respective first-stage dual constraint.

Then for each scenario we remove assignment between job j' and its mate (line 16), and update dual variable $\beta_{j'k}$ (line 17), which is necessary to keep the respective constraints (3.16) satisfied. Consequently, for each scenario there is only one agent-job assignment missing and the current labeling, i.e., the values of dual variables $(\alpha,\ \beta)$, is feasible for (3.16). Thus, one iteration of HM (line 18) is sufficient to achieve an optimal labeling for updated weights $\hat{q}^k_{i'j'}$ for each scenario. This procedure (lines 4-18) is performed until for every pair (i, j) we have

$$\sum_{k=1}^{K} (\alpha_{ik} + \beta_{jk}) \;\geq\; w_{ij}, \quad \forall i, j\,. \tag{3.23}$$

Therefore, the following result holds.

Proposition 12. *Let A and J be a pair of subsets of agents and jobs, respectively, such that $|A| = |J|$. Then after line 18 of Algorithm 2,*

$$\sum_{k} \left[\sum_{i \in A} \alpha_{ik} + \sum_{j \in J} \beta_{jk} \right] \geq W_1[A, J]. \tag{3.24}$$

Proof. Follows directly from (3.23). □

This result implies that after the above described procedure (lines 4-18), obtained assignment satisfies the necessary optimality condition for all closed subsets in the second stage. In other words, for all closed subsets in the first stage, the respective aggregated first-stage dual constraints are satisfied. On the other hand, we do not consider the subsets of agents and jobs that are not closed in the first stage since constructing assignments between exponentially many possible subsets of agents and jobs is prohibitively expensive. Therefore, contrary to the case for EGA-I, EGA-II does not check subsets with cardinality strictly greater than one.

Next consider an assignment $(i, mate_k[i])$ in the k^{th} scenario. If $\hat{q}^k_{i,mate_k[i]}$ is an updated weight, and we have the same assignment in all scenarios, i.e., $(i, mate_k[i])$ is a closed subset in the second stage, we can move this assignment to the first stage without changing the total weight of the current assignments and without affecting other assignments. If we can do this for all such pairs, then we have an optimal solution due to the strong duality. However, it may not be the case that each time we have the same assignment $(i, mate[i])$ in all scenarios as the original problem is NP-hard. Therefore, moving this assignment to the first stage changes assignments in the scenarios, where i is not assigned to $mate[i]$.

On the other hand, keeping assignment $(i, mate[i])$, for the subsets that are not closed in the second stage indicates that we have an updated arc weight which actually does not exist and we can not find an assignment corresponding to it (i.e., primal infeasible). EGA-II decreases the weight of the assignment for such pairs to their original values and updates the assignments across all scenarios (lines 19-26) to accommodate for this change. Finally, the remaining agent-job pairs with the updated weights are moved to the first stage and a separate assignment problem is solved for them.

Lemma 2. *Let Z_2^{EGA} be the weight of the assignment returned by EGA-II. Then*

$$Z_2^{EGA} \geq Z_2^{GA} . \tag{3.25}$$

Proof. Denote by $(\tilde{\alpha}, \tilde{\beta})$ the dual second-stage myopic solutions. Consider labeling $(\hat{\alpha}, \hat{\beta})$ obtained after line 26 of Algorithm 5. Since the weight updates only increase the arc weights, we have

$$\sum_{k=1}^{K}\sum_{i=1}^{n}\hat{\alpha}_{ik} + \sum_{k=1}^{K}\sum_{j=1}^{n}\hat{\beta}_{jk} \geq \sum_{k=1}^{K}\sum_{i=1}^{n}\tilde{\alpha}_{ik} + \sum_{k=1}^{K}\sum_{j=1}^{n}\tilde{\beta}_{jk}.$$

Observe that the procedure after line 26 can only potentially improve the weight of the final assignment returned by EGA-II, which concludes the proof. □

Theorem 4. *Approximation bounds given for GAE in Theorems 2 and 3 are valid for EGA, and its solution satisfies*

$$Z^{EGA} \geq Z^{GAE} \geq Z^{GA}.$$

Proof. The necessary result directly follows from Lemmata 1 and 2. □

Proposition 13. *Time complexity of EGA is* $O(Kn^4)$.

Proof. First, we consider complexity of EGA-I. It is easy to observe that lines 1-4 take $O(Kn^3)$ time. Next step is to find closed subsets. There can be at most n closed subsets and $\mathrm{O}(Kn)$ time is required to construct each of them. Thus, in total $\mathrm{O}(Kn^2)$ time is needed. Next, solving the assignment problem for a closed subset (line 10) takes $O(|A|^3)$, which is bounded by $O(|A|n^2)$. Since $\sum_{\{A,J\}\in C}|A| \leq n$, the total complexity of solving assignment problems for all closed subsets requires $O(n^3)$ time. The complexity of sorting in line 12 is $O(|A|\lg|A|)$ and similar to our previous argument, total time complexity of sorting is $O(n \lg n)$. If one starts with $t = 1$ and tries each index sequentially, the complexity of lines 13-16 for all closed subsets is $O(n)$. Consequently, complexity of updating the weights is $O(n^3)$. At most n iterations of HM is performed between lines 20-25; thus, the resetting procedure is of $O(n^3)$ time complexity. Finally, solving $K+1$ assignment problems (line 28) requires $O(Kn^3)$. Consequently, the total time complexity of EGA-I is $\mathrm{O}(Kn^3)$.

The most time consuming procedure for EGA-II is updating weights and assignments (lines 4-18). Since there are n^2 first-stage dual constraints, the outer loop requires $\mathrm{O}(n^2)$ operations. The inner loop is to increase cardinality of assignments by 1 across all scenarios, in the worst case. Since each stage of HM requires $\mathrm{O}(n^2)$ time, the complexity of the inner loop is at most $\mathrm{O}(Kn^2)$. Thus, the total time complexity of EGA-II is $\mathrm{O}(Kn^4)$. □

3.4.3 Improving EGA with Local Search (EGA LS)

In this section we introduce a greedy local exchange based heuristic that seeks to further improve the results obtained by EGA-I and EGA-II. Let $(X, Y) = (x, y_1, y_2, \ldots, y_K)$ be a feasible assignment for the 2SSLA problem. Here, we distinguish between an assignment (i, j) and a pair $[i, j]$. The former one indicates that agent i is assigned to job j whereas for the latter one we do not imply any dependence. We say that pair $[i, j]$ belongs to the partial solution X (or Y) if assignments $(i, mate[i])$ and $(mate[j], j)$ are both at the first stage (or second stage). Define the following neighborhoods for a given solution (X, Y):

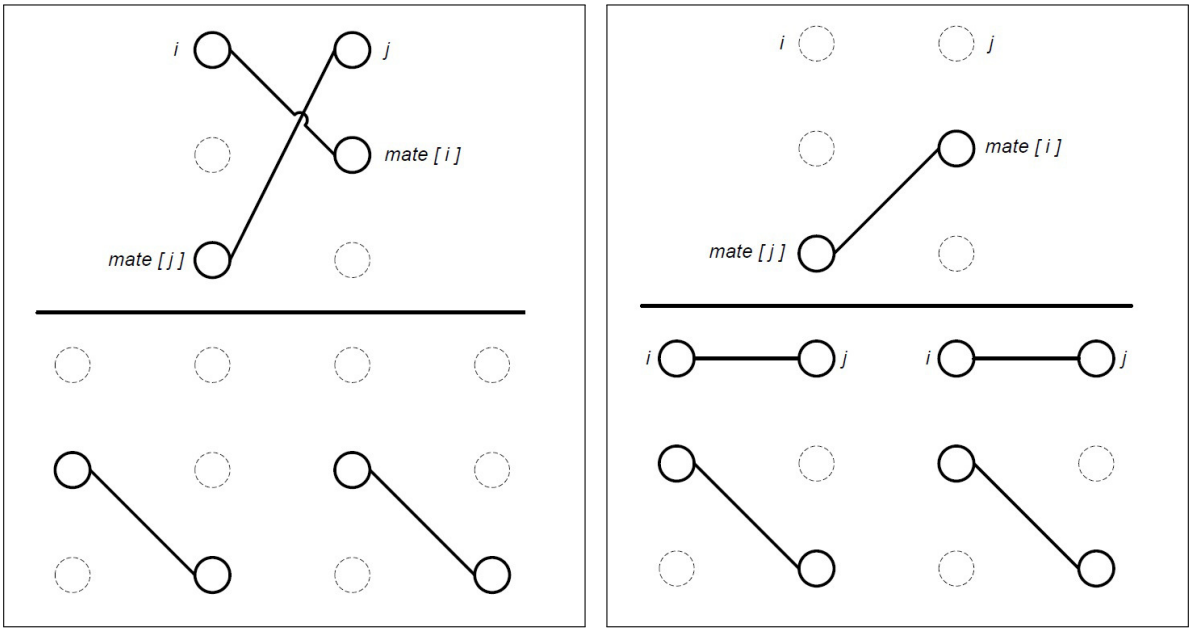
Algorithm 5: `EGA-II`

Input: n agents, n jobs, K scenarios, w_{ij}, q_{ij}^k, p_k

Let $\tilde{q}_{ij}^k = p_k q_{ij}^k$ and G_k be the graph for the k^{th} scenario in the second-stage
foreach *Scenario* k **do**
    Run `Hungarian Method` on G_k to find an assignment for the k^{th} scenario
while *There is an* i *and* j *such that* $[w_{ij} > \sum_k (\alpha_{ik} + \beta_{jk})]$ **do**
    Let $(i', j') = \arg\max \{w_{ij} - \sum_k (\alpha_{ik} + \beta_{jk})\}$
    Add (i', j') to set $\mathbf{R}$
    begin Update assignments on $G_k \; \forall \; k$
        Let $s_k = \alpha_{i'k} + \beta_{j'k} - \tilde{q}_{i'j'}^k$ `// Slack for the` k^{th} `scenario`
        if $\sum_k s_k > 0$ **then**
            foreach *Scenario* k **do**
                $\tilde{q}_{i'j'}^k := \tilde{q}_{i'j'}^k + \left(w_{i'j'} - \sum_k \tilde{q}_{i'j'}^k\right) \left(\frac{s_k}{\sum_k s_k}\right)$
        else
            foreach *Scenario* k **do**
                $\tilde{q}_{i'j'}^k := w_{i'j'} \left(\frac{\tilde{q}_{i'j'}^k}{\sum_k \tilde{q}_{i'j'}^k}\right)$
        foreach *Scenario* k **do**
            Remove edge $(mate\,[j'], j)$ from assignment on G_k
            $\beta_{j'k} = \max_i \left\{\tilde{q}_{ij'}^k - \alpha_{ik}\right\}$
            Perform one iteration of *Hungarian Method* on G_k
begin Reset all G_k
    while *There is a pair* $\boldsymbol{(i,j)} \in \boldsymbol{R}$ *such that* $\boldsymbol{mate}[\boldsymbol{i}]$ *is not* $\boldsymbol{j} \; \forall \; k$ **do**
        foreach *Scenario* k **do**
            Reset edge weight $\tilde{q}_{ij}^k$ to its original value
            if $mate[i] = j$ **then**
                Remove assignment (i, j) from G_k
                Perform one iteration of *Hungarian Method* on G_k
        Remove the pair (i, j) from $\mathbf{R}$
Let G_0 be the graph for the first-stage. Move all pairs $(i, j) \in \mathbf{R}$ from G_k, $\forall \; k$, to G_0
Run `Hungarian Method` on G_0
$Z_2^{EGA} = \sum_{i \in G_0} w_{i,mate[i]} + \sum_k \sum_{i \in G_k} \tilde{q}_{i,mate_k[i]}^k$
return Z_2^{EGA}, G_0, *and* $G_k \; \forall \; k$

- Neighborhood N_1 for solution (X, Y) is defined to be the set of all solutions obtained by moving any pair $[i, j]$ from X to Y. This implicitly requires $[i, j] \in X$. Thus, to maintain feasibility, if a solution $(\overline{X}, \overline{Y}) \in N_1(X, Y)$, then we have $(mate[j], mate[i]) \in \overline{X}$ and $(i, j) \in \overline{Y}$, assuming that $(\overline{X}, \overline{Y})$ is obtained from (X, Y) with respect to pair $[i, j]$. This exchange process is illustrated in Figure 24.

- Neighborhood N_2 for solution (X, Y) is defined to be the set of all solutions obtained by moving any pair $[i, j]$ from Y to X. This implicitly requires $[i, j] \in Y$. Thus, to maintain feasibility, if a solution $(\overline{X}, \overline{Y}) \in N_2(X, Y)$, then we have $(i, j) \in \overline{X}$ and $(mate[j], mate[i]) \in \overline{Y}$. This exchange process is illustrated in Figure 25.

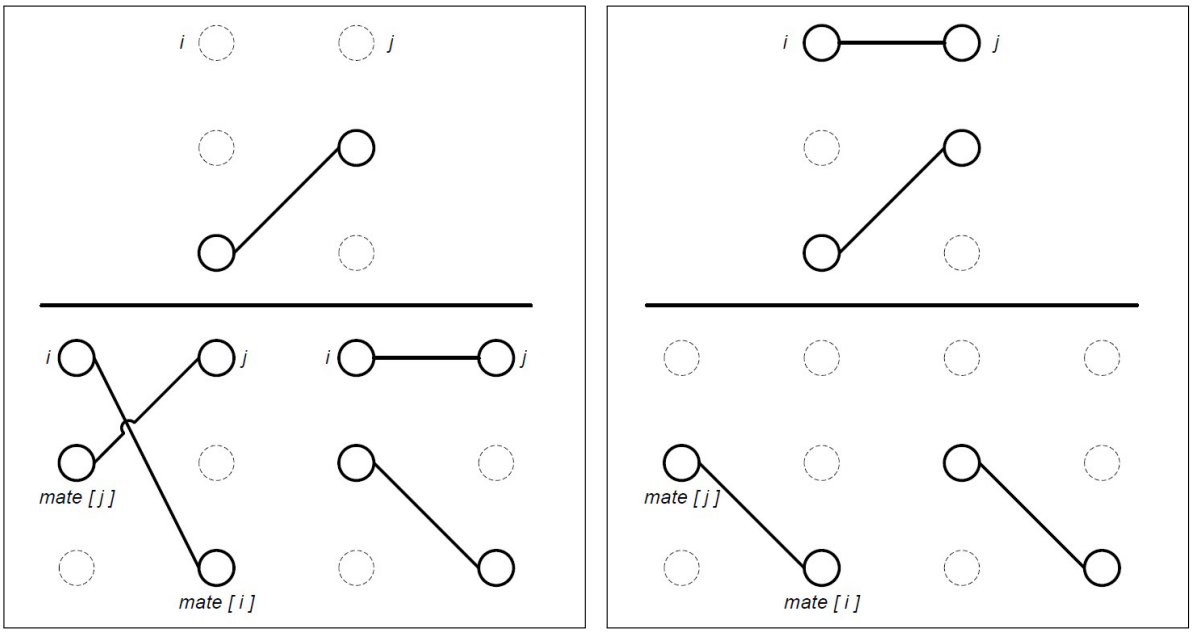


(a) Initial solution.

(b) Final solution obtained.

Figure 24: The neighborhood N_1.

Proposition 14. *The solutions obtained by GAE and EGA-I are locally optimal with respect to the neighborhood N_1.*



(a) Initial solution. (b) Final solution obtained.

Figure 25: The neighborhood N_2.

Proof. Let (X, Y) be the solution returned after line 1 of Algorithm 4. Consider a pair $[i, j] \in X$. We check whether there is a better solution $(\overline{X}, \overline{Y}) \in N_1(X, Y)$ with respect to $[i, j]$ as follows:

$$\sum_{k=1}^{K} \tilde{q}_{ij}^{k} > w_{i,mate[i]} + w_{mate[j],j} - w_{mate[j],mate[i]}. \tag{3.26}$$

Assume that $[i, j]$ satisfies (3.26) and consider the respective dual solution for X. Then we know that the constraints (3.6) are tight for $(i, mate[i])$ and $(mate[j], j)$:

$$\alpha_i + \beta_{mate[i]} = w_{i,mate[i]}; \quad \alpha_{mate[j]} + \beta_j = w_{mate[j],j}; \quad \text{and} \quad \alpha_{mate[j]} + \beta_{mate[i]} \geq w_{mate[j],mate[i]}.$$

Then, from (3.26), we have

$$\begin{aligned} \sum_{k=1}^{K} \tilde{q}_{ij}^{k} \; &> \; w_{i,mate[i]} + w_{mate[j],j} - w_{mate[j],mate[i]} \\ &\geq \; (\alpha_i + \beta_{mate[i]}) \; + \; (\alpha_{mate[j]} + \beta_j) \; - \; (\alpha_{mate[j]} + \beta_{mate[i]}) \\ &= \; \alpha_i \; + \; \beta_j. \end{aligned}$$

However, this is not possible because the necessary optimality condition for sets with unit cardinality implies that $\alpha_i + \beta_j \geq w_{ij} \geq \sum_{k=1}^{K} \tilde{q}_{ij}^k$ due to the update in (3.17). Therefore, the necessary result follows. □

Next consider EGA-II. Let (X, Y) be the solution obtained by EGA-II and $[i, j] \in Y$. We check whether switching to a solution in the neighborhood N_2 of (X, Y) with respect to $[i, j]$ improves the current solution. Formally, we verify whether

$$w_{ij} > \sum_{k=1}^{K} \left(\tilde{q}_{i,mate[i]}^k + \tilde{q}_{mate[j],j}^k - \tilde{q}_{mate[j],mate[i]}^k \right). \tag{3.27}$$

If (3.27) is satisfied, then removing pair $[i, j]$ for all scenarios in the second stage and assigning i to j in the first stage improves the current solution. This process is illustrated in Figure 25. However, we may further improve our new solution by running one iteration of Hungarian Method for the first stage and K iterations of Hungarian Method for the second stage. Since each iteration of Hungarian Method requires $O(n^2)$ time, this update requires $O(Kn^2)$ time. Furthermore, at most n pairs may be moved to the first stage, which results in a total time complexity of $O(Kn^3)$ for local search after EGA-II.

3.4.4 Analytical Observations

Next, we discuss performance of GA, GAE, and EGA on two carefully constructed classes of test instances. Assume that in both classes, we have $2n$ agents, $2n$ jobs, and K scenarios, $K \leq n$. We partition the set of all agents and jobs into two groups: G_1 and G_2, where G_1 contains the first n agents and n jobs and G_2 contains remaining n agents and n jobs. Now we describe two types of instances.

Split Instances: We construct this type of instances as follows:

$$w_{ij} = \begin{cases} 1 & \text{for} \quad (i,j) \in G_1, \\ 2K & \text{for} \quad (i,j) \in G_2 \text{ and } i = j, \\ 0 & \text{o/w.} \end{cases}$$

$$q_{ij}^k = \begin{cases} K & \text{for} \quad (i,j) \in G_1 \text{ and } i+k-1 \equiv j \mod n, \\ K & \text{for} \quad (i,j) \in G_2, \\ 0 & \text{o/w.} \end{cases}$$

and $p_k = 1/K, \;\; k = 1, \ldots, K$. The structure of these instances is illustrated in Figure 26. It is optimal to make assignments for G_2 in the first stage while for G_1 it is optimal to make assignments in the second stage. Therefore, the total weight of the optimal assignment is $3nK$.

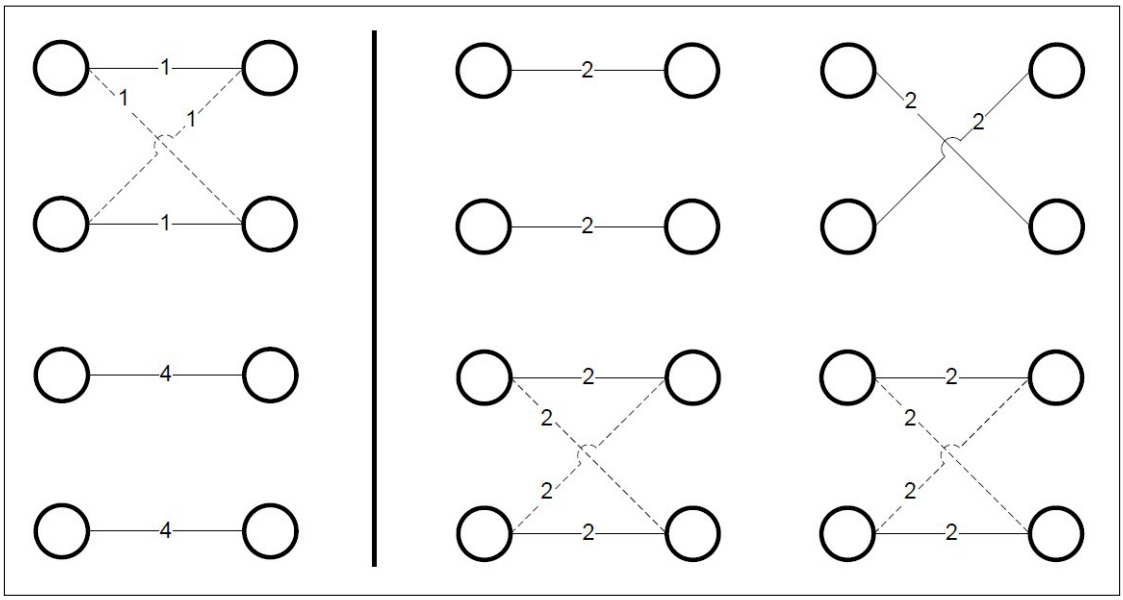


Figure 26: A *split instance* for $K = n = 2$ (only nonzero arcs are shown). Thick lines show first-stage and second-stage myopic solutions.

Interleaved Instances: We construct this type of instances as follows:

$$w_{ij} = \begin{cases} 1 & \text{for} \quad (i,j) \in G_1, \\ 2K & \text{for} \quad (i,j) \in G_2 \text{ and } i = j, \\ 0 & \text{o/w.} \end{cases}$$

$$q_{ij}^k = \begin{cases} K & \text{for} \quad (i,j) \in G_1 \text{ and } i+k-1 \equiv j \mod n, \\ K & \text{for} \quad i \in G_1,\ j \in G_2, \text{ and } i+k-1 \equiv j-n \mod n, \\ K & \text{for} \quad i \in G_2,\ j \in G_1, \text{ and } i+k-1 \equiv j \mod n, \\ 0 & \text{o/w.} \end{cases}$$

and $p_k = 1/K, \quad k = 1, \ldots, K$. The structure of these instances is illustrated in Figure 27. The optimal solution should have all assignments within G_1 in the second stage and all assignments within G_2 in the first stage, with the total weight of $3nK$.

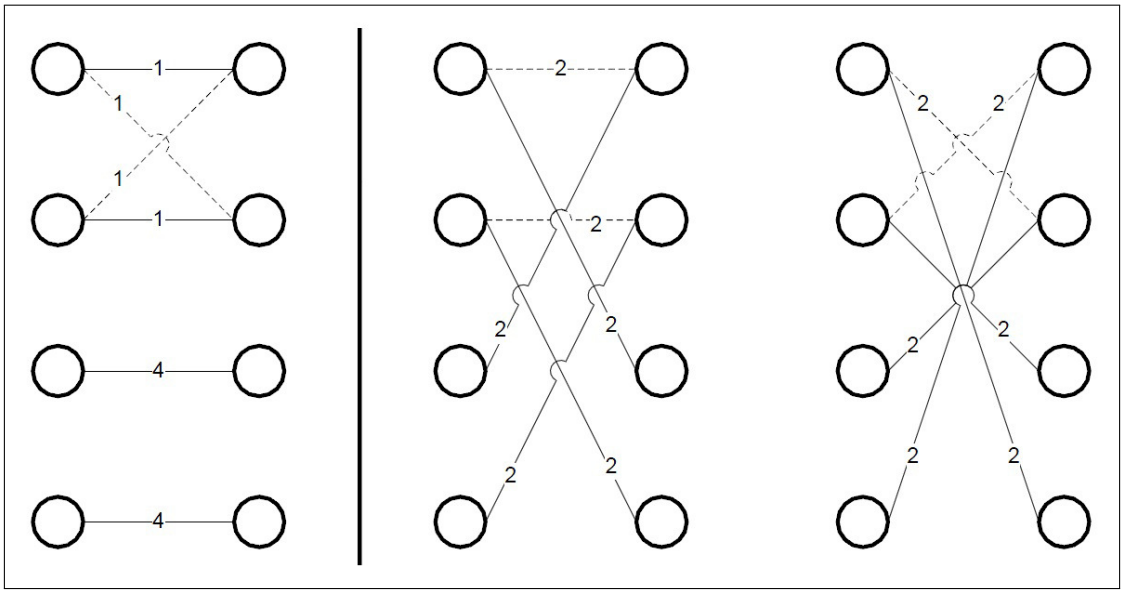


Figure 27: An *interleaved instance* for $K = n = 2$ (only nonzero arcs are shown). Thick lines show first-stage and second-stage myopic solutions.

Proposition 15. *The weights of the assignments obtained by GA, GAE, and EGA for split instances are* $n(2K+1)$, $n(2K+1)$, *and* $3nK$, *respectively.*

Proof. It is easy to check that GA would make all assignments in the first stage and the total weight of this assignment would be $n(2K+1)$. Next we consider GAE. From Figure 26, it is clear that the first-stage myopic solution satisfies the necessary optimality condition for the sets of unit cardinality. Thus, the solution returned by GAE is the same as the solution returned by GA.

Finally, we consider EGA. Notice that the total weight of the assignments within G_1 in the first stage is n whereas the expected total weight of assignments within G_1 in the second stage is nK. Thus, the first-stage myopic solution violates the necessary optimality condition. Since G_1 is a closed subset in the second-stage myopic solution, EGA-I moves all assignments to the second stage. Therefore, the weight of the assignment returned by EGA-I is at least $3nK$. Since the weight of the optimal assignment for 'split' instances is $3nK$ and EGA returns the best of EGA-I and EGA-II, EGA finds the optimal solution. □

Proposition 16. *The weights of the assignments obtained by GA, GAE, and EGA for interleaved instances are $n(2K+1)$, $n(2K+1)$, and $3nK$, respectively.*

Proof. Similar to the proof of Proposition 15. □

Concluding this section, we want to emphasize the importance of the constructed problem instances. They demonstrate that GA and GAE may provide results that are significantly worse than the optimal solution, while EGA returns the optimal solution in both cases.

3.5 COMPUTATIONAL EXPERIMENTS

3.5.1 Setup

Five classes of test instances are considered in our computational study. To provide the comparison of GAE and EGA, we set the first two classes to be similar to those used in [50].

Uncorrelated Instances: All edge weights are drawn from $N(10, 15)$, the normal distribution with mean 10 and standard deviation 15, i.e.,

$$w_{ij} \sim N(10, 15),\ \forall i, j, \qquad\qquad q_{ij}^{k} \sim N(10, 15),\ \forall i, j, k.$$

If the generated weight is negative, then it is set to zero. All scenarios have the same probability.

Correlated Instances: For these instances, the second-stage weights are correlated. Specifically,

$$w_{ij} \sim N(10, 15),\ \forall i, j, \qquad q_{ij} \sim N(10, 15),\ \forall i, j, \qquad q_{ij}^k \sim q_{ij} + N(0, 5),\ \forall i, j, k.$$

If the generated weight is negative, then it is set to zero. All scenarios have the same probability. One can easily conclude that if the weight q_{ij} generated for edge (i, j) is low (high), then the weight of edge (i, j) will be low (high) in all scenarios.

The intuition behind the next class of test instances is to have the necessary optimality condition satisfied for nearly all unit cardinality sets, but possibly violated for subsets with two agents and two jobs. As it is demonstrated later, both GA and GAE fail to identify such rather simple weight dependencies.

Pairwise-Correlated Instances: Unlike *correlated* instances, the correlation in these instances is not just on a single agent and job, but on pairs of agents and jobs. Specifically, we use

$$w_{ij} = \begin{cases} N(200, 40) & \text{for} \quad i \equiv 0 \mod 3, \\ N(140, 30) & \text{for} \quad i \equiv 1 \mod 3 \text{ and } i \leq j \leq i+1, \\ N(140, 30) & \text{for} \quad i \equiv 2 \mod 3 \text{ and } i-1 \leq j \leq i, \\ N(10, 15) & \text{o/w.} \end{cases}$$

$$q_{ij}^k = \begin{cases} N(200, 40) & \text{for} \quad k \equiv 0 \mod 2,\ i \equiv 1 \mod 3,\ \text{and } j = i, \\ N(200, 40) & \text{for} \quad k \equiv 1 \mod 2,\ i \equiv 1 \mod 3,\ \text{and } j = i+1, \\ N(200, 40) & \text{for} \quad k \equiv 0 \mod 2,\ i \equiv 2 \mod 3,\ \text{and } j = i, \\ N(200, 40) & \text{for} \quad k \equiv 1 \mod 2,\ i \equiv 2 \mod 3,\ \text{and } j = i-1, \\ N(10, 15) & \text{o/w.} \end{cases}$$

Next we provide details for *split-like* and *interleaved-like* instances. Both classes are based on the instances introduced in Section 3.4.4 with some modifications that aim at "randomizing" their structure. In particular, we add a third class of agents and jobs to the 'split' and 'interleaved' instances with uniformly generated assignment weights. Thus, there are $3n$ agents, $3n$ jobs, and K scenarios, $K \leq n$. Define G_1, G_2, and G_3 to be the respective sets, each with n agents and n jobs.

Split-like Instances: Let

$$w_{ij} = \begin{cases} U[900/K, 1000/K] & \text{for} \quad (i,j) \in G_1, \\ U[500, 1000] & \text{for} \quad (i,j) \in G_2, \\ U[100, 1000] & \text{for} \quad (i,j) \in G_3, \\ U[2, 10] & \text{o/w.} \end{cases}$$

$$q_{ij}^k = \begin{cases} U[800, 900] & \text{for} \quad (i,j) \in G_1 \text{ and } \; i+k-1 \equiv j \mod n, \\ U[100, 500] & \text{for} \quad (i,j) \in G_2, \\ U[100, 1000] & \text{for} \quad (i,j) \in G_3, \\ U[2, 10] & \text{o/w.} \end{cases}$$

Interleaved-like Instances: Let

$$w_{ij} = \begin{cases} U[900/K, 1000/K] & \text{for} \quad (i,j) \in G_1, \\ U[4000, 5000] & \text{for} \quad (i,j) \in G_2 \text{ and } \; i = j, \\ U[100, 1000] & \text{for} \quad (i,j) \in G_3, \\ U[2, 10] & \text{o/w.} \end{cases}$$

$$q_{ij}^k = \begin{cases} U[800, 900] & \text{for} \quad (i,j) \in G_1 \text{ and } \; i+k-1 \equiv j \mod n, \\ U[1500, 2000] & \text{for} \quad i \in G_1, \; j \in G_2, \text{ and } \; i+k-1 \equiv j-n \mod n, \\ U[1500, 2000] & \text{for} \quad i \in G_2, \; j \in G_1, \text{ and } \; i+k-1 \equiv j \mod n, \\ U[100, 1000] & \text{for} \quad (i,j) \in G_3, \\ U[2, 10] & \text{o/w.} \end{cases}$$

The probability for each scenario is set to be $1/K$ for each instance from either class.

In our computational experiments, we use CPLEX 12 [74]. The algorithms are coded in C++ and implemented on a Windows XP based machine with Intel Xeon 3 GHz processor and 3GB RAM. In our experiments we consider problems with 2, 3, 5, 10, and 20 scenarios and 10, 20, 50, 100, and 200 agents/jobs. For each of these 25 configurations, we conduct 10 replications and report their averages as well as standard deviations.

3.5.2 Results and Discussion

We report statistics for CPLEX solver as well as GA, GAE, and EGA algorithms. The first two columns in all tables are self-explanatory and report the sizes of test instances. We provide average running time (in seconds) for CPLEX in *Time* column. As we enforce the time limit of 3600 seconds for CPLEX, an average running time of 3600 seconds in this column implies that CPLEX is unable to solve all IP formulations to optimality. In such cases we use the LP relaxation solution for comparison purposes. The next four columns report the percentage difference from the CPLEX solution for GA-I and GA-II, its standard deviation for GA and the overall running time for GA (in seconds). Next we provide the percentage difference of the solution returned only by GAE-I from the CPLEX solution (recall that GAE-II is the same as GA-II). We also report the respective standard deviation and the combined time of GAE-I and GAE-II. The next four columns correspond to the results obtained by EGA. Finally, we provide results for EGA-II with the local search procedure including the standard deviation and the time spent for performing the local search (excluding the time for obtaining initial EGA-II solution). The best results are shown in bold.

Results for the *uncorrelated* instances are given in Table 6. It can be observed that CPLEX has difficulty in solving large instances whereas the running time does not exceed 5 seconds for all other algorithms. As expected, EGA finds the best results in all cases. EGA solution yields a significant improvement over GA solution and is reasonably better than GAE solution. EGA-II (with and without local search) is successful in improving the second-stage myopic solution.

Table 7 summarizes the results for the *correlated* instances. Both EGA-I and EGA-II find nearly optimal solutions and local search further improves the solution of EGA-II. GAE-I also performs very well on these instances, which is expected due to the structure of these test instances. Note that if an edge has a large weight in one scenario, then it should have a large weight in all scenarios. Thus, most of the time, it is better to make assignments for the agents and jobs incident to such edges in the second stage. Since GAE checks the necessary optimality condition for subsets of unit cardinality, its success on these instances is expected.

Results for the *pairwise-correlated* instances are summarized in Table 8. Both GA and GAE perform rather poorly. On the other hand, EGA (especially EGA-I) is successful in detecting correlation between pairs of agents and jobs. Since correlation is between pairs of jobs but not for larger subsets, the local search procedure also performs well.

Results for the *split-like* instances are summarized in Table 9. One can observe that CPLEX runs out of time for larger instances. Solutions found by GA and GAE are poor. In fact, GA and GAE find almost the same solutions. On the other hand, solution of EGA-I is only 1% worse than the CPLEX solution on average. EGA-II performs rather poorly for large instances; however the local search procedure is able to eliminate this deviation as shown in EGA-II LS. The reason that EGA-I is successful on this class of instances is that it is the only algorithm that can move the whole set G_1 of agents and jobs to the second stage as this set does not satisfy the necessary optimality condition.

Table 10 reports results for the *interleaved-like* instances. This time EGA-II is the best among all algorithms considered. Its solution is only about 0.3% worse than the CPLEX solution. Since EGA-II is already very successful, additional improvement from the local search heuristic is not expected. It is also interesting to observe that contrary to EGA, the first-stage solution is better than the second-stage solution with GA and GAE.

In summary, we should point out that the results for the *pairwise-correlated* as well as *split-like* and *interleaved-like* instances indicate that if the weight dependencies between subsets of agents and jobs become more complicated (e.g., in comparison with the *correlated* instances) then GA and GAE algorithms fail to correctly identify the proper assignments between such subsets of agents and jobs. This is due to the fact that these algorithms check the necessary optimality conditions only for subsets of size 1 and n. On the other hand, EGA is specifically designed to locate some of these subsets of agents and jobs, which significantly improves the quality of the obtained solutions.

Table 6: Results for uncorrelated instances.

		CPLEX	GA				GAE			EGA				EGA LS		
Scenarios	**Agents**	**Time**	**I (%)**	**II (%)**	**Sdev (%)**	**Time (s)**	**I (%)**	**Sdev (%)**	**Time (s)**	**I (%)**	**II (%)**	**Sdev (%)**	**Time (s)**	**II LS (%)**	**Sdev (%)**	**Time (s)**
	10	0	9.60	10.24	2.58	0	**2.00**	1.87	0	**2.00**	6.36	1.65	0	3.72	1.71	0
	20	0	10.03	9.15	2.61	0	5.33	1.42	0	5.33	4.36	1.20	0	**2.71**	1.29	0
2	**50**	0	6.88	6.95	1.42	0	4.55	0.99	0	4.55	3.88	1.19	0	**3.22**	1.03	0
	100	1	6.24	6.28	0.54	0	4.80	0.67	0	4.80	3.23	0.87	0	**3.08**	0.76	0
	200	10	5.16	5.36	0.70	0	4.53	0.64	0	4.50	2.70	0.60	0	**2.67**	0.57	0
	10	0	9.33	8.84	4.36	0	**3.48**	2.24	0	**3.48**	6.83	1.84	0	3.84	1.19	0
	20	0	6.87	8.05	1.89	0	**3.77**	1.76	0	**3.77**	6.15	1.85	0	4.96	1.76	0
3	**50**	0	6.34	6.65	0.98	0	5.65	0.93	0	5.65	5.42	1.06	0	**4.79**	0.74	0
	100	7	5.37	5.10	0.67	0	5.03	0.67	0	4.83	3.65	0.47	0	**3.42**	0.54	0
	200	325	4.65	4.74	0.52	0	4.55	0.49	0	4.21	3.13	0.46	0	**3.09**	0.50	0
	10	0	12.11	8.82	2.71	0	6.85	2.70	0	5.68	8.51	2.63	0	**2.63**	2.33	0
	20	0	7.85	6.74	1.28	0	6.63	1.52	0	5.71	6.55	1.51	0	**5.48**	1.56	0
5	**50**	1	5.25	5.53	1.12	0	5.10	1.03	0	**3.97**	4.66	0.97	0	4.40	0.89	0
	100	119	4.22	4.10	0.71	0	4.20	0.71	0	3.51	3.55	0.67	0	**3.36**	0.64	0
	200	2113	3.98	3.68	0.37	0	3.98	0.36	0	3.51	3.06	0.48	2	**3.02**	0.49	0
	10	0	8.46	8.16	2.11	0	6.21	1.79	0	5.65	7.51	1.57	0	**2.53**	1.35	0
	20	0	4.42	6.56	1.91	0	4.26	1.98	0	**3.96**	6.56	1.98	0	4.81	1.81	0
10	**50**	26	4.63	4.81	1.21	0	4.58	1.21	0	**3.74**	4.40	1.16	0	4.07	1.26	0
	100	1536	4.26	3.58	0.68	0	4.26	0.68	0	**3.31**	3.41	0.48	0	3.40	0.48	0
	200	3600	3.73	3.49	0.55	0	3.73	0.55	0	3.40	2.93	0.41	3	**2.82**	0.36	0
	10	0	4.47	9.87	3.15	0	**2.34**	2.05	0	**2.34**	9.87	2.05	0	4.39	1.90	0
	20	0	5.86	4.39	1.56	0	5.47	1.49	0	4.50	4.39	1.49	0	**3.92**	1.10	0
20	**50**	96	4.42	4.03	1.32	0	4.42	1.32	0	**3.31**	4.03	1.32	0	3.57	1.20	0
	100	2391	3.81	3.80	0.34	0	3.81	0.34	0	**3.32**	3.71	0.34	0	3.71	0.34	0
	200	3600	0.43	0.76	0.32	1	0.43	0.32	1	0.23	0.52	0.18	5	**0.20**	0.00	0

Table 7: Results for correlated instances.

		CPLEX	GA				GAE			EGA				EGA LS		
Scenarios	**Agents**	**Time**	**I (%)**	**II (%)**	**Sdev (%)**	**Time (s)**	**I (%)**	**Sdev (%)**	**Time (s)**	**I (%)**	**II (%)**	**Sdev (%)**	**Time (s)**	**II LS (%)**	**Sdev (%)**	**Time (s)**
	10	0	6.89	2.49	0.93	0	0.32	0.33	0	0.19	0.13	0.15	0	**0.09**	0.11	0
	20	0	4.65	1.90	0.78	0	0.56	0.51	0	0.55	0.13	0.16	0	**0.12**	0.16	0
2	**50**	0	4.98	1.18	0.36	0	0.61	0.20	0	0.60	0.23	0.17	0	**0.20**	0.14	0
	100	0	4.80	1.05	0.18	0	0.73	0.17	0	0.72	0.15	0.16	0	**0.13**	0.16	0
	200	2	4.76	0.82	0.14	0	0.81	0.09	0	0.80	0.16	0.09	0	**0.15**	0.08	0
	10	0	5.65	1.70	1.20	0	0.73	0.32	0	0.31	0.18	0.00	0	**0.15**	0.00	0
	20	0	4.86	1.24	0.93	0	0.69	0.42	0	0.57	0.19	0.17	0	**0.17**	0.17	0
3	**50**	0	4.78	0.67	0.24	0	1.00	0.26	0	1.00	0.15	0.10	0	**0.15**	0.10	0
	100	0	4.54	0.66	0.15	0	1.18	0.15	0	1.18	0.22	0.08	0	**0.21**	0.08	0
	200	3	4.64	0.46	0.12	0	1.22	0.12	0	1.22	0.15	0.07	0	**0.14**	0.07	0
	10	0	4.57	1.06	0.47	0	1.13	0.47	0	0.95	0.41	0.26	0	**0.27**	0.24	0
	20	0	4.20	0.65	0.52	0	1.20	0.41	0	0.87	0.16	0.11	0	**0.15**	0.11	0
5	**50**	0	4.27	0.45	0.15	0	1.43	0.15	0	1.43	0.19	0.10	0	**0.15**	0.08	0
	100	0	4.60	0.32	0.14	0	1.61	0.14	0	1.59	0.15	0.07	0	**0.14**	0.07	0
	200	5	4.48	0.24	0.06	0	1.63	0.06	0	1.63	0.09	0.04	0	**0.09**	0.03	0
	10	0	4.26	0.50	0.35	0	1.17	0.31	0	1.17	0.14	0.23	0	**0.13**	0.23	0
	20	0	4.12	0.46	0.46	0	1.84	0.34	0	1.42	0.28	0.32	0	**0.20**	0.31	0
10	**50**	0	4.05	0.25	0.19	0	1.97	0.19	0	1.97	0.13	0.13	0	**0.11**	0.09	0
	100	1	4.23	0.20	0.08	0	2.02	0.08	0	1.82	0.10	0.07	0	**0.08**	0.07	0
	200	10	4.22	0.14	0.05	0	2.03	0.05	0	1.38	0.07	0.03	1	**0.07**	0.04	0
	10	0	3.44	0.28	0.28	0	1.71	0.26	0	0.80	0.10	0.07	0	**0.07**	0.06	0
	20	0	3.90	0.19	0.14	0	1.91	0.14	0	1.18	0.10	0.11	0	**0.07**	0.07	0
20	**50**	0	4.07	0.10	0.05	0	2.21	0.05	0	1.33	0.06	0.04	0	**0.05**	0.04	0
	100	3	4.00	0.11	0.06	0	2.23	0.06	0	1.51	0.07	0.04	0	**0.06**	0.04	0
	200	2526	4.12	0.06	0.01	1	2.39	0.01	1	0.52	**0.04**	0.01	2	**0.04**	0.01	0

Table 8: Results for pairwise-correlated instances.

		CPLEX	GA				GAE			EGA				EGA LS		
Scenarios	**Agents**	**Time**	**I (%)**	**II (%)**	**Sdev (%)**	**Time (s)**	**I (%)**	**Sdev (%)**	**Time (s)**	**I (%)**	**II (%)**	**Sdev (%)**	**Time (s)**	**II LS (%)**	**Sdev (%)**	**Time (s)**
	10	0	11.35	36.30	5.00	0	10.91	4.69	0	3.43	**0.00**	0.00	0	**0.00**	0.00	0
	20	0	14.14	31.56	2.86	0	13.29	3.63	0	3.04	0.44	0.00	0	**0.00**	0.00	0
2	**50**	0	15.31	28.55	1.55	0	14.51	1.24	0	4.21	1.16	1.95	0	**0.00**	0.00	0
	100	0	16.34	27.45	1.70	0	15.13	1.90	0	4.55	0.42	1.32	0	**0.00**	0.00	0
	200	1	16.38	26.36	0.88	0	14.92	0.75	0	4.49	0.63	1.09	0	**0.00**	0.00	0
	10	0	13.87	35.07	2.62	0	9.62	2.25	0	3.29	**0.00**	0.00	0	**0.00**	0.00	0
	20	0	14.65	30.45	2.19	0	11.18	1.95	0	4.46	3.48	2.60	0	**0.00**	0.00	0
3	**50**	0	14.66	28.54	1.57	0	10.55	1.09	0	3.56	4.02	1.99	0	**0.00**	0.00	0
	100	0	16.08	27.41	0.97	0	11.20	0.93	0	5.53	2.37	2.84	0	**0.00**	0.01	0
	200	1	15.66	26.12	0.90	0	11.04	1.10	0	4.88	0.60	1.26	0	**0.00**	0.00	0
	10	0	14.57	35.08	3.06	0	13.72	2.47	0	5.54	**0.00**	0.00	0	**0.00**	0.00	0
	20	0	14.87	30.51	2.55	0	12.86	2.43	0	3.75	5.23	1.97	0	**0.00**	0.00	0
5	**50**	0	15.46	28.32	1.72	0	14.22	1.33	0	4.12	3.65	1.67	0	**0.00**	0.00	0
	100	0	15.42	27.57	1.27	0	13.88	0.68	0	4.19	3.84	2.80	0	**0.01**	0.04	0
	200	2	15.97	26.31	0.92	0	13.83	1.11	0	4.25	4.76	2.47	0	**0.00**	0.00	0
	10	0	14.87	36.65	2.65	0	14.71	2.49	0	2.32	**0.00**	0.00	0	**0.00**	0.00	0
	20	0	13.14	30.83	2.24	0	12.96	2.11	0	0.97	14.76	0.86	0	**0.00**	0.00	0
10	**50**	0	14.82	28.53	1.29	0	14.57	1.15	0	2.26	18.68	1.61	0	**0.00**	0.00	0
	100	1	15.72	27.79	1.21	0	15.29	1.10	0	2.85	9.22	1.67	0	**0.00**	0.00	0
	200	5	15.60	26.66	0.88	0	15.16	0.74	0	2.41	8.70	1.57	1	**0.01**	0.02	0
	10	0	14.87	35.35	2.13	0	14.70	2.15	0	2.99	3.92	0.00	0	**0.00**	0.00	0
	20	0	16.04	30.10	1.84	0	15.48	1.74	0	4.28	8.80	3.20	0	**0.00**	0.00	0
20	**50**	0	15.67	28.71	1.63	0	15.00	1.42	0	3.25	14.11	1.95	0	**0.00**	0.00	0
	100	2	16.17	27.28	0.95	0	15.88	1.03	0	2.61	19.12	1.26	0	**0.00**	0.00	0
	200	2524	15.91	26.18	0.64	1	15.50	0.48	1	2.57	15.70	1.38	3	**0.00**	0.00	0

Table 9: Results for split-like instances.

		CPLEX	GA				GAE			EGA				EGA LS		
Scenarios	**Agents**	**Time**	**I (%)**	**II (%)**	**Sdev (%)**	**Time (s)**	**I (%)**	**Sdev (%)**	**Time (s)**	**I (%)**	**II (%)**	**Sdev (%)**	**Time (s)**	**II LS (%)**	**Sdev (%)**	**Time (s)**
2	**10**	0	22.01	8.41	2.99	0	15.39	2.99	0	3.17	1.75	1.32	0	**0.85**	0.99	0
	20	0	17.23	11.89	1.27	0	13.89	0.99	0	2.00	3.66	0.75	0	**1.03**	0.79	0
	50	0	14.41	15.60	0.62	0	13.30	0.36	0	1.20	6.93	0.32	0	**1.04**	0.25	0
	100	0	13.63	16.73	0.26	0	13.19	0.15	0	**0.73**	6.22	0.15	0	0.78	0.09	0
	200	5	12.73	17.12	0.10	0	12.57	0.09	0	**0.33**	6.71	0.07	0	0.51	0.07	0
3	**10**	0	25.43	4.01	3.13	0	21.20	3.13	0	2.67	0.94	0.72	0	**0.41**	0.47	0
	20	0	19.99	10.12	1.44	0	18.75	1.44	0	**1.12**	5.78	1.17	0	1.50	0.67	0
	50	0	19.04	14.62	0.54	0	18.85	0.54	0	**0.71**	9.69	0.37	0	1.35	0.35	0
	100	1	18.75	16.22	0.20	0	18.71	0.20	0	**0.33**	9.28	0.09	0	0.98	0.09	0
	200	27	18.32	16.84	0.07	0	18.32	0.07	0	**0.18**	7.94	0.03	0	0.51	0.03	0
5	**20**	0	23.33	9.29	1.19	0	22.70	1.19	0	**0.37**	7.81	0.35	0	2.10	0.35	0
	50	0	23.27	14.43	0.29	0	23.21	0.29	0	**0.43**	11.84	0.33	0	1.73	0.31	0
	100	5	23.38	16.19	0.17	0	23.37	0.17	0	**0.22**	11.02	0.08	0	0.77	0.08	0
	200	860	22.91	16.79	0.04	0	22.91	0.04	0	**0.11**	10.18	0.03	1	0.48	0.03	0
10	**50**	2	26.98	14.19	0.32	0	26.97	0.32	0	**0.25**	13.50	0.22	0	0.58	0.15	0
	100	120	26.77	16.10	0.14	0	26.77	0.14	0	**0.07**	13.97	0.06	0	0.41	0.06	0
	200	3600	26.41	16.80	0.03	0	26.41	0.03	0	**0.11**	12.86	0.03	2	0.34	0.03	0
20	**100**	2892	28.52	16.20	0.12	0	28.52	0.12	0	**0.16**	15.84	0.10	0	0.45	0.10	0
	200	3600	28.18	16.77	0.03	1	28.18	0.03	1	**0.10**	15.38	0.02	5	0.27	0.02	0

Table 10: Results for interleaved-like instances.

		CPLEX	GA				GAE			EGA				EGA LS		
Scenarios	**Agents**	**Time**	**I (%)**	**II (%)**	**Sdev (%)**	**Time (s)**	**I (%)**	**Sdev (%)**	**Time (s)**	**I (%)**	**II (%)**	**Sdev (%)**	**Time (s)**	**II LS (%)**	**Sdev (%)**	**Time (s)**
2	**10**	0	6.55	30.21	0.88	0	5.67	0.36	0	5.67	0.57	0.73	0	**0.30**	0.36	0
	20	0	6.44	29.45	0.67	0	5.68	0.26	0	5.68	0.69	0.64	0	**0.50**	0.44	0
	50	0	5.92	29.57	0.15	0	5.76	0.10	0	5.75	0.32	0.13	0	**0.28**	0.12	0
	100	0	5.77	29.35	0.08	0	5.70	0.08	0	5.70	0.17	0.05	0	**0.17**	0.05	0
	200	3	5.62	29.30	0.06	0	5.60	0.06	0	5.60	0.09	0.01	0	**0.09**	0.01	0
3	**10**	0	9.66	30.69	1.61	0	8.59	0.45	0	8.53	0.86	0.87	0	**0.22**	0.39	0
	20	0	9.01	29.31	0.73	0	8.40	0.25	0	8.32	0.81	0.41	0	**0.62**	0.39	0
	50	0	8.57	28.99	0.16	0	8.45	0.13	0	8.42	0.28	0.07	0	**0.26**	0.06	0
	100	1	8.34	29.24	0.09	0	8.32	0.10	0	8.31	0.19	0.04	0	**0.18**	0.04	0
	200	12	8.20	29.36	0.07	0	8.20	0.06	0	8.19	**0.09**	0.01	0	**0.09**	0.01	0
5	**20**	0	10.72	29.29	0.54	0	10.36	0.36	0	10.36	0.69	0.52	0	**0.54**	0.40	0
	50	0	10.35	29.51	0.20	0	10.32	0.20	0	10.24	**0.21**	0.08	0	**0.21**	0.08	0
	100	4	10.38	29.43	0.15	0	10.37	0.14	0	10.33	**0.14**	0.05	0	**0.14**	0.05	0
	200	984	10.24	29.40	0.05	0	10.24	0.05	0	10.22	**0.06**	0.02	0	**0.06**	0.02	0
10	**50**	1	12.01	28.75	0.21	0	11.99	0.22	0	11.90	**0.14**	0.06	0	**0.14**	0.06	0
	100	65	11.94	29.11	0.08	0	11.94	0.08	0	11.90	**0.09**	0.04	0	**0.09**	0.04	0
	200	3600	11.83	29.19	0.07	0	11.83	0.07	0	11.82	**0.07**	0.01	1	**0.07**	0.01	0
20	**100**	3273	12.75	28.96	0.13	0	12.75	0.13	0	12.74	**0.11**	0.02	0	**0.11**	0.02	0
	200	3600	12.62	29.12	0.06	1	12.62	0.06	1	12.61	**0.06**	0.01	3	**0.06**	0.01	0

3.6 CONCLUSION

This chapter discusses several greedy approximation algorithms for the 2SSLA problem. The proposed necessary optimality condition unifies two recent greedy approximation algorithms from the literature. It is further used in the development of a more advanced algorithmic approach. While EGA preserves the approximation guarantees of GAE, we are not able to prove whether EGA provides a strictly better approximation bound. However, analytical observations and computational results indicate that EGA has strictly better results on some rather broad classes of the two-stage stochastic linear assignment problem.

As future research directions, one can use the proposed necessary optimality condition to develop new algorithms with better approximation guarantees, consider the extension to the multi-stage stochastic linear assignment problem, or focus on the problems with stochastic right-hand sides, e.g., when multiple jobs can be performed by the same agent. Furthermore, the results of the reported computational experiments indicate that the integrality gap is very small for most of the considered test instances. Thus, development of approximation algorithms based on the LP relaxation of the original integer program is among other promising research directions.

3.7 ACKNOWLEDGMENTS

Content of this chapter is reproduced from [78] with permission of Taylor & Francis.

4.0 ON SPEED SCALING VIA INTEGER PROGRAMMING

4.1 INTRODUCTION

Electricity cost is the major budget item for most data warehouses and computing centers. Google engineers predict that energy cost is very likely to overtake the hardware cost in the near future [20]. Energy efficiency is also a key concern for various types of battery-powered remote devices. For these reasons, major chip manufacturers such as Intel, AMD and IBM produce microprocessors that can run at variable speed. Thus, in order to minimize the total energy consumption, modern operating systems need not only to schedule processes but also to determine at which speed should the processors run.

Dynamic speed scaling literature, which can be traced back to [116], is focused on two broad problem settings:

- Jobs have deadlines and the goal is to find feasible schedules minimizing energy consumption [14, 16, 116]. Algorithms for a processor with sleep state under this problem setting also exist [68, 75].
- Jobs do not have deadlines and besides minimizing energy consumption, the goal is to find schedules that optimize some scheduling metric, e.g., average flow time [5, 15, 17, 38, 85, 84].

All these studies target the single processor case, which is a difficult problem by itself, whereas the multiprocessor case has not attracted much attention, primarily due to its complexity. Jobs with deadline on identical processors is studied by [4, 6, 23], while [86] focuses on the no-deadline scenario. Nevertheless, heterogeneous processors are becoming more common and require special treatment [28]. Authors of [64, 65] consider heterogeneous processors without

sleep states and no deadlines for jobs. We refer the reader to [3] for a more thorough review of the speed scaling literature.

The focus of our work is a relatively high-level problem setting that includes heterogeneous parallel processors with sleep states. Given a processing requirement, our goal is to select a subset of processors and distribute the load over these processors so as to minimize the total energy consumption. Specifically, consider a data warehouse with a total of n processors (or a CPU with n cores). Each processor can be either turned off or set to run at some positive continuous speed s (i.e., frequency). If processor i runs at speed $s_i > 0$, it consumes fixed γ_i (to turn the processor on) and variable $\alpha_i s^{\beta_i}$ energy. Given service load L (e.g., number of CPU cycles) to be satisfied by the warehouse, the speed scaling problem is to decide which processors to turn on and determine their speeds in order to minimize the total energy consumed.

Without loss of generality, assume M_i ($L \geq M_i$) is an upper bound on the speed of processor i and let the binary variable x_i denote whether processor i is turned on. Given nonnegative parameters α_i, β_i, and γ_i for all i, speed scaling with convex power functions (SSCPF) can be modeled as the following mixed integer nonlinear program (MINLP):

$$
\begin{aligned}
\text{SSCPF:} \qquad \min_{x,s} \quad & \sum_{i=1}^{n} (\alpha_i s_i^{\beta_i} + \gamma_i x_i) \\
\text{s.t.} \quad & \sum_{i=1}^{n} s_i \geq L \\
& M_i x_i \geq s_i, \ \forall i \\
& s_i \geq 0, \ x_i \in \{0,1\}, \ \forall i
\end{aligned}
$$

Solving convex MINLPs is challenging as they generalize the linear integer programming, known to be NP-hard [56]. Relaxing integrality restrictions results in a convex optimization problem. Thus, most of the exact solution approaches focus on using branch-and-bound (B&B) framework where subproblems are either convex programs or their linear approximations. Duran and Grossmann [48] introduced the outer approximation (OA) algorithm, which was further improved by Fletcher and Leyffer [53].

OA algorithm constructs lower linear approximations to the nonlinear functions via tangent lines. At each iteration, first, integer optimal solution to the current relaxation is obtained through B&B. Then, using a nonlinear program solver, the optimal fractional solution of the convex program that corresponds to the current integral solution is found. Finally, the relaxation is updated by including approximation of the nonlinear functions using this optimal fractional solution and the procedure is repeated. An optimal solution of the original problem is found if the same integer solution is encountered as each integer solution provides an upper bound while the current approximation is a lower bound [48]. If we assume that all integer variables are bounded, then the number of integer solutions is finite. Thus, OA algorithm is finite.

Rather than solving each relaxation to optimality, Quesada and Grossmann [100] provide a more efficient implementation of the algorithm where only one search tree is kept and the relaxation is updated as new integer solutions are found within a B&B framework. In contrast to OA, the extended cutting plane (ECP) method proposed by Westerlund and Pettersson [115] approximates nonlinear functions using values of the fractional variables obtained from relaxation, and thus ECP does not solve any nonlinear programs. Method is finite for any constant precision level $\epsilon > 0$. However, to achieve the required precision, a large number of cuts (approximations) may be needed, which is going to be a burden on the linear program (LP) solver, especially when cuts are added globally and there are many functions to be approximated. A thorough review of the MINLP literature is beyond the scope of this work and we refer the reader to [26, 27].

In our problem setting, once an integral feasible solution is encountered, finding optimal speed levels for the processors is known as the continuous nonlinear resource allocation problem (NRAP) [73] in the literature. NRAP is essentially equivalent to the continuous nonlinear knapsack problem (NKP). The book by Ibaraki and Katoh [73] provides a detailed treatment of NRAP and its variants. Most of the algorithms devised for the convex, separable nonlinear case are based on the Karush-Kuhn-Tucker (KKT) optimality conditions and use of ϵ-accuracy notion [30, 31, 71, 72]. Integer NKP (i.e., the case when speed levels are required to be integer) is beyond the scope of this study and interested readers may consult [31, 39].

One problem that is somewhat similar to SSCPF is the *Convex Quadratic Transportation-Cost Uncapacitated Facility Location Problem* studied in [63]. The authors derive KKT solutions to the *distribution problem* of a single customer given a set of facilities that are open. The major difference of their problem is that the nonlinear structure of the transportation cost function studied there is dependent on the customer but not facilities. To make a comparison, NRAP is the problem with many customers and a single facility whereas the distribution problem studied in [63] has many facilities but a single customer. Furthermore, their work is purely motivated on strengthening the continuous relaxation of the problem.

To the best of our knowledge, the class of NRAPs represented by SSCPF has not been studied in the past. In Section 4.2, based on KKT conditions, we discuss a line search method to find optimal speed values for a given subset of processors. In Section 4.3, we identify several polynomially solvable cases and show that the problem is NP-hard even when $\beta_i = 2$ for all i. Next, an approximation algorithm for SSCPF is provided in Section 4.4. Section 4.5 describes a pseudo-polynomial time dynamic programming algorithm (DP) for a special case of the problem. Subsequently , the DP is converted into a *fully polynomial-time approximation scheme* (FPTAS). For the general case, we provide an outer approximation implementation in Section 4.6. Section 4.7 summarizes our computational experiments.

4.2 OPTIMALITY CONDITIONS

Consider a set $F \subseteq \{1, \ldots, n\}$ of processors and assume that all processors belonging to this set are turned on, i.e., $x_i = 1$ iff $i \in F$. Then, SSCPF reduces to finding optimal speed values for processors in F:

$$\min_{s} \quad \sum_{i \in F} \alpha_i s_i^{\beta_i} \tag{4.1}$$

$$\text{s.t.} \quad \sum_{i \in F} s_i \geq L \tag{4.2}$$

$$s_i \geq 0 \quad \forall i \in F \tag{4.3}$$

Note that in (4.1), we disregard the term $\sum_{i\in F}\gamma_i$ as it is constant for fixed F. Let λ and μ_i be the dual variables corresponding to the constraints (4.2) and (4.3), respectively. Then we can write KKT conditions as follows

$$\sum_{i\in F} s_i - L \geq 0\,, \qquad s_i \geq 0 \quad \forall i \in F \tag{4.4}$$

$$\lambda \geq 0\,, \qquad \mu_i \geq 0 \quad \forall i \in F \tag{4.5}$$

$$\lambda\left(\sum_{i\in F} s_i - L\right) = 0\,, \qquad \mu_i s_i = 0 \quad \forall i \in F \tag{4.6}$$

$$\nabla\left(\sum_{i\in F}\alpha_i s_i^{\beta_i}\right) - \nabla\left(\lambda\left(\sum_{i\in F} s_i - L\right)\right) - \nabla\left(\sum_{i\in F}\mu_i s_i\right) = 0 \tag{4.7}$$

One can show that (4.7) is equivalent to

$$\alpha_i\beta_i s_i^{\beta_i-1} - \lambda - \mu_i = 0 \qquad \forall i \in F \tag{4.8}$$

In general, if F is the set of processors that are turned on, the optimal speed values for processors in F can be found by using the following procedure:

- set $\lambda > 0$ and $\mu_i = 0$ (using (4.6)) for all $i \in F$,
- solve the following equation in λ (e.g., using a standard line search method [44]):

$$L = \sum_{i\in F} \sqrt[\beta_i-1]{\frac{\lambda}{\alpha_i\beta_i}} \tag{4.9}$$

- using (4.8), set

$$s_i = \sqrt[\beta_i-1]{\frac{\lambda}{\alpha_i\beta_i}} \qquad \forall i \in F \tag{4.10}$$

Therefore, finding the optimal speed values for a given set of processors F essentially reduces to line search, which is key to our solution approach. Next, using this solution approach, we identify two polynomially solvable cases and show that SSCPF is NP-hard even when $\beta_i = 2$ for all i.

4.3 POLYNOMIALLY SOLVABLE CASES AND COMPLEXITY

For SSCPF, each processor is characterized by three parameters: α, β, and γ. As we show below, computational complexity of the problem depends on variation of these parameters between processors.

Proposition 17. *Whenever at least two of the parameters $\{\alpha, \beta, \gamma\}$ are the same for all processors (e.g., $\alpha_i = \alpha$ and $\beta_i = \beta$ for all i), SSCPF is solvable in polynomial time.*

Proof. The key observation is that if the number of processors used in an optimal solution were known, say k, then we could solve the problem easily. Specifically, one sorts processors in increasing order of the parameter that is not fixed and then selects the first k of them. The obtained solution is optimal due to the following observation. Whenever a solution prefers a processor j, $k < j \leq n$, to a processor i, $1 \leq i \leq k$, it can be improved by replacing processor j with processor i even without changing the speed values (assuming that the value of the parameter that varies is strictly smaller for i). Thus, the problem can be solved by simply checking each possible value of k, $1 \leq k \leq n$. □

Proposition 18. *If $\beta_i = 1$ for all i, then the optimal solution consists of the single processor*

$$i^* \in \arg\min_{1 \leq i \leq n} \{\alpha_i L + \gamma_i\}$$

Proof. This specific case of SSCPF is actually an instance of UFL problem with single customer [63]. Nevertheless, we give a constructive proof here. Assume that a subset F of processors is selected and turned on. Then (4.8) reduces to

$$\alpha_i - \lambda - \mu_i = 0 \qquad \forall i \in F$$

A KKT solution for F is given by

$$i^* \in \arg\min_{i \in F} \{\alpha_i\}, \quad \lambda = \alpha_{i^*}, \quad \mu_i = \alpha_i - \lambda \quad \forall i \in F, \quad s_{i^*} = L, \quad s_i = 0 \quad \forall i \in F \setminus \{i^*\}$$

Hence, it is optimal to turn on the processor with smallest α value. As it is always optimal to turn on a single processor, it is clear that the one resulting with smallest energy consumption should be turned on. □

Theorem 5. *Problem SSCPF is NP-hard even if $\beta_i = 2$ for all i.*

Proof. Consider a set of processors F to be turned on. If $\beta_i = 2$, $\forall i$, then from (4.10) we have

$$\begin{aligned} & \alpha_j s_j = \alpha_i s_i \quad \forall i, j \in F \\ & \frac{s_j}{s_i} = \frac{\alpha_i}{\alpha_j} \quad \forall i, j \in F \qquad (4.11) \\ \Rightarrow \quad & \frac{\sum_{j \in F} s_j}{s_i} = \sum_{j \in F} \frac{\alpha_i}{\alpha_j} \quad \forall i \in F \\ \Rightarrow \quad & s_i = \frac{L}{\sum_{j \in F} \frac{\alpha_i}{\alpha_j}} = \frac{L}{\alpha_i} \frac{1}{\sum_{j \in F} \frac{1}{\alpha_j}}, \qquad \forall i \in F \qquad (4.12) \end{aligned}$$

If we rewrite the objective function for SSCPF when processors in the set F are turned on:

$$\begin{aligned} \sum_{i=1}^{n} (\alpha_i s_i^{\beta_i} + \gamma_i x_i) &= \sum_{i \in F} \alpha_i s_i^2 + \sum \gamma_i \\ &= \sum_i \alpha_i \left(\frac{L}{\alpha_i} \frac{1}{\sum_j \frac{1}{\alpha_j}} \right)^2 + \sum \gamma_i \\ &= \frac{L^2}{\left(\sum_j \frac{1}{\alpha_j} \right)^2} \sum_i \alpha_i \frac{1}{\alpha_i^2} + \sum \gamma_i \\ &= \frac{L^2}{\left(\sum_j \frac{1}{\alpha_j} \right)^2} \sum_i \frac{1}{\alpha_i} + \sum \gamma_i \\ &= L^2 \frac{1}{\sum_i \frac{1}{\alpha_i}} + \sum \gamma_i \qquad (4.13) \end{aligned}$$

One can show that, if we let $\alpha_i = \frac{1}{\gamma_i}$ $\forall i$, then (4.13) reduces to

$$L^2 \frac{1}{\sum \gamma_i} + \sum \gamma_i$$

Since the set F can be chosen in many ways, let's define a new variable $t = \sum \gamma_i$ and consider the function

$$g(t) = L^2 \frac{1}{t} + t$$

One can also show that

$$\text{if } t > 0 \text{ then } g(t) \geq 2L \text{ and}$$
$$t = L \quad \text{iff } \; g(t) = 2L$$

and, therefore, $2L$ is a lower bound on objective function value in this special case and this lower bound can be achieved iff there exists a subset of processors F which satisfies $\sum \gamma_i = L$.

Now we can make a reduction from the SUBSET SUM [56] problem to prove that the general case of our problem is NP-hard. Consider an instance of the subset sum problem defined over a set of integers $R = \{r_1, r_2, ..., r_n\}$ with n items and a required sum of L. Now, we generate an instance of the speed scaling problem with n parallel processors as follows: let $\beta_i = 2$, $\forall i$, set $\gamma_i = r_i$, let $\alpha_i = \frac{1}{\gamma_i}$, and consider a required load of L. Then in the light of our previous discussion, there exists a subset of R with a sum of L iff the constructed SSCPF instance has an optimal objective value of $2L$. □

Theoretical computational complexity of the other two cases when (1) $\alpha_i = \alpha$ for all i and (2) $\gamma_i = \gamma$ remain open problems.

4.4 A GREEDY APPROXIMATION ALGORITHM

Consider a subset of processors $T \subseteq N = \{1, 2, \ldots, n\}$. We define the *variable* energy consumption of T to be $E(T) = \min\left\{ \sum_{j \in T} \alpha_j s_j^{\beta} \;\middle|\; \sum_{j \in T} s_j = L, s_j \geq 0 \right\}$ and the *fixed* energy consumption of T to be $\Gamma(T) = \sum_{j \in T} \gamma_j$. Thus, total energy consumption using processors in T is equal to $E(T) + \Gamma(T)$. Using this notation, Algorithm 6 provides the pseudocode for a greedy heuristic which we show to be an approximation algorithm.

Algorithm 6: Greedy Algorithm (GA)

1 Sort processors in increasing order of their fixed cost γ.

2 Let T_{ij} be the set of processors $\{i, \ldots, j\}$, $1 \leq i \leq j \leq n$.

3 Return T_{ij} with smallest $E(T_{ij}) + \Gamma(T_{ij})$.

Proposition 19. *GA is an n-factor approximation algorithm for SSCPF.*

Proof. Let T^* be the set of processors selected by an optimal solution where $i, j \in T^*$ are the indices of the processors with smallest and largest γ values, respectively. Observe that $E(T_{ij}) \leq E(T^*)$ as $T^* \subseteq T_{ij}$. Furthermore, $\Gamma(T_{ij}) \leq (j - i + 1)\gamma_j \leq n\Gamma(T^*)$. Hence, the solution of GA satisfies $Z^a \leq E(T_{ij}) + \Gamma(T_{ij}) \leq nZ^*$. □

GA has a time complexity of $O(n^2)$. It can be shown that the bound provided by GA is strict when optimal solution includes only the processor with smallest γ value and the processor with the largest γ value.

4.5 FPTAS FOR SPECIAL CASE

In this section we consider the special case of SSCPF where β_i is the same (i.e., $\beta_i = \beta$) for all $i = 1, \ldots, n$.

4.5.1 A Dynamic Programming Approach

Let $E(\cdot)$ and $\Gamma(\cdot)$ be defined as in previous section. Define $T[i, \gamma]$ to denote a subset of processors $\{1, 2, \ldots, i\}$ minimizing $E(\cdot)$ with a total fixed cost of $\Gamma(\cdot) = \gamma$. If no such subset exists, then we let $T[i, \gamma] = \emptyset$ where $E(\emptyset) = \infty$. Initializing $T[0, \gamma] = \emptyset$ for $\gamma \in \{0, 1, \ldots\}$, all $T[i, \gamma]$, $i \in N$, can be found via the following recursion:

$$T[i,\gamma] = \begin{cases} \{i\} \cup T[i-1, \gamma - \gamma_i] & \text{if} \quad E\left(\{i\} \cup T[i-1, \gamma - \gamma_i]\right) < E\left(T[i-1, \gamma]\right) \\ T[i-1, \gamma] & \text{o/w} \end{cases}$$

An optimal solution to the problem is given by $T[n, \gamma^*]$, $\gamma^* \in \arg\min\left\{\gamma + E\big(T[n, \gamma]\big)\right\}$.

Proposition 20. *The dynamic programming algorithm given above is correct when $\beta_i = \beta$ for all $i = 1, \ldots, n$.*

Proof. We prove by induction on i. Indeed $T[0,\gamma] = \emptyset$. Assume for $i-1$, $T[i-1,\gamma]$ is the set minimizing $E(\cdot)$. If i does not belong to the subset of processors from $\{1,2,\ldots,i\}$ that minimizes energy consumption, then it must hold that $T[i,\gamma] = T[i-1,\gamma]$. Otherwise, let $\tilde{T}[i-1,\gamma] \neq T[i-1,\gamma]$ be another subset of the processors $\{1,2,\ldots,i-1\}$. By induction hypothesis, $E(T[i-1,\gamma]) \leq E(\tilde{T}[i-1,\gamma])$. We show that $E(\{i\} \cup T[i-1,\gamma]) \leq E(\{i\} \cup \tilde{T}[i-1,\gamma])$. One can verify that,

$$\lambda = \frac{\beta L^{\beta-1}}{\left(\sum_{j\in T}\left(\frac{1}{\alpha_j}\right)^{\frac{1}{\beta-1}}\right)^{\beta-1}}, \quad s_j = \left(\frac{\lambda}{\alpha_j\beta}\right)^{\frac{1}{\beta-1}} \quad \forall j \in T, \quad E(T) = \frac{L^{\beta}}{\left(\sum_{j\in T}\left(\frac{1}{\alpha_j}\right)^{\frac{1}{\beta-1}}\right)^{\beta-1}}$$

is a KKT solution of the system defined by $E(T)$ for any subset T. If $E(T) \leq E(\tilde{T})$, then it follows that

$$\sum_{j\in T}\left(\frac{1}{\alpha_j}\right)^{\frac{1}{\beta-1}} \geq \sum_{j\in \tilde{T}}\left(\frac{1}{\alpha_j}\right)^{\frac{1}{\beta-1}} \quad \Rightarrow \quad \left(\frac{1}{\alpha_i}\right)^{\frac{1}{\beta-1}} + \sum_{j\in T}\left(\frac{1}{\alpha_j}\right)^{\frac{1}{\beta-1}} \geq \left(\frac{1}{\alpha_i}\right)^{\frac{1}{\beta-1}} + \sum_{j\in \tilde{T}}\left(\frac{1}{\alpha_j}\right)^{\frac{1}{\beta-1}}$$

Hence, $E(\{i\} \cup T[i-1,\gamma-\gamma_i]) \leq E(\{i\} \cup \tilde{T}[i-1,\gamma-\gamma_i])$ for all $\tilde{T}$. □

The running time of the dynamic programming algorithm is $O(n\overline{\gamma})$, where $\overline{\gamma} = \sum_{j=1}^{n}\gamma_j$ is a simple upper bound on γ. Note that for any T, $\{i\} \cup T$ can be correctly compared to T in constant time if the sum $\sum_{j\in T}(1/\alpha_j)^{1/(\beta-1)}$ is readily available. One can construct a counterexample to show that DP does not work when β_i are allowed to be different. Nevertheless, DP can be used as a heuristic when β_i is not the same for all i.

4.5.2 FPTAS

To obtain a FPTAS, we follow the approach used for the knapsack problem [79, 113], and scale γ_i values. We let $\hat{\gamma}_j = \left\lceil \frac{\gamma_j}{K} \right\rceil$, which is equivalent to performing recursion on $\gamma \in \{0, K, 2K, 3K, \ldots\}$. Furthermore, we modify DP to output a set $T[n,\gamma^*]$ where

$$\gamma^* \in \arg\min\left\{\gamma + \frac{E(T[n,\gamma])}{K}\right\}$$

Let T and T^* be the optimal set of processors for the scaled and the original instances, respectively. Then the following relations hold:

$$Z = E(T) + \sum_{j \in T} \gamma_j \leq K \left(\frac{E(T)}{K} + \sum_{j \in T} \left\lceil \frac{\gamma_i}{K} \right\rceil \right) \leq$$
$$K \left(\frac{E(T^*)}{K} + \sum_{j \in T^*} \left\lceil \frac{\gamma_i}{K} \right\rceil \right) = E(T^*) + K \sum_{j \in T^*} \left\lceil \frac{\gamma_i}{K} \right\rceil \leq$$
$$E(T^*) + K \sum_{j \in T^*} \left(\frac{\gamma_i}{K} + 1 \right) = E(T^*) + \sum_{j \in T^*} \gamma_i + K|T^*| = Z^* + K|T^*|$$

For solution of the DP to be $\epsilon-$approximate:

$$\frac{Z - Z^*}{Z^*} \leq \epsilon \quad \Rightarrow \quad \frac{K|T^*|}{Z^*} \leq \epsilon \quad \Rightarrow \quad K \leq \epsilon \frac{Z^*}{|T^*|}$$

Algorithm 7 provides the pseudocode of a FPTAS for speed scaling problem.

Algorithm 7: FPTAS

1 Use GA to get $Z^a \leq nZ^*$. Let $K = \epsilon \frac{Z^a}{n^2}$.

2 Scale the fixed costs: $\hat{\gamma}_i = \left\lceil \frac{\gamma_i}{K} \right\rceil$.

3 Run DP on the scaled instance.

Proposition 21. *FPTAS outputs an ϵ-approximate solution to SSCPF in $O\left(\frac{n^3}{\epsilon}\right)$ time when $\beta_i = \beta$, $\forall i$.*

Proof. GA provides a solution with $Z^* \leq Z^a \leq nZ^*$. Thus, the maximum fixed cost that the DP algorithm has to look up is Z^a, which results in a time complexity of $O(nZ^a)$. Setting $K = \epsilon \frac{Z^a}{n} \frac{1}{n} \leq \epsilon Z^* \frac{1}{|T^*|}$, the optimal solution to the scaled instance provides an ϵ-approximate solution to the original instance. Thus,

$$O\left(n \left\lceil \frac{Z^a}{K} \right\rceil \right) = O\left(nZ^a \frac{n^2}{\epsilon Z^a} \right) = O\left(\frac{n^3}{\epsilon} \right)$$

□

4.6 AN OUTER APPROXIMATION ALGORITHM

In this section we follow notation from [26]. Consider the generic MINLP:

$$\text{MINLP:} \qquad z_{\texttt{MINLP}} = \min \Big\{ f(x,y) \mid g_j(x,y) \leq 0 \ \forall j, x \in X \cap \mathbb{Z}^n, y \in Y \Big\}$$

where $X \subseteq \mathbb{R}^n$ and $Y \subseteq \mathbb{R}^p$ are polyhedral sets. Outer approximation is applicable when functions $f : X \times Y \to \mathbb{R}$ and $g : X \times Y \to \mathbb{R}^m$ are convex and continuously differentiable. Thus, relaxing integrality constraints results in a convex program that can be efficiently solved. Since f and g are convex and differentiable, they can be linearly approximated from below at any point $(x_o, y_o) \in X \times Y$ using the first order term of their Taylor series expansions around (x_o, y_o). Now assume that $\mathcal{H}$ is the set of all (x, y) such that $y \in Y$ is an optimal solution to MINLP for each $x \in X \cap \mathbb{Z}^n$. Then consider the resulting relaxation of MINLP:

$$\begin{aligned} \text{MINLP-OA:} \quad z_{\texttt{MINLP-OA}} = \min \Bigg\{ z \;\Bigg|\; & \nabla f(x^k, y^k)^T \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} \leq z \ \forall (x^k, y^k) \in \mathcal{H}, \\ & \nabla g_j(x^k, y^k)^T \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} \leq 0 \quad \forall j, \quad \forall (x^k, y^k) \in \mathcal{H}, \\ & x \in X \cap \mathbb{Z}^n, \quad y \in Y, \quad z \in \mathbb{R} \qquad \Bigg\} \end{aligned}$$

Theorem 6 ([48]). *If MINLP is feasible, f and g are convex and differentiable, and constraint qualification holds for each $(x^k, y^k) \in \mathcal{H}$, then $z_{\texttt{MINLP}} = z_{\texttt{MINLP-OA}}$ and every optimal solution of MINLP is also an optimal solution of MINLP-OA.*

Since $\mathcal{H}$ is potentially huge, the successful implementation starts with a relaxation of MINLP-OA written only for a small subset $H \subseteq \mathcal{H}$ and adds new *cuts* within a branch-and-cut framework as necessary. We follow this approach in our work. Our functions are seperable and smooth, and thus we explicitly give the outer approximation of SSCPF for a subset of linearization points $H_i \subseteq \mathcal{H}_i$ for each processor i:

$$
\begin{aligned}
\text{SSCPF-OA:} \qquad \min_{x} \quad & \sum_{i=1}^{n} (f_i + \gamma_i x_i) \\
\text{s.t.} \quad & \sum_{i=1}^{n} s_i \geq L \\
& M_i x_i \geq s_i \quad \forall i \\
& f_i \geq \alpha_i \beta_i s_h^{\beta_i - 1} s - \alpha_i (\beta_i - 1) s_h^{\beta_i} \quad \forall s_h \in H_i, \ \forall i \\
& s_i \geq 0, \ f_i \geq 0, \ x_i \in \{0, 1\} \quad \forall i
\end{aligned}
$$

Since the number of integral solutions to SSCPF is finite and we have to approximate functions only at integral solutions, the outer approximation formulation proposed is finite. Furthermore, we do not need a convex program solver as the optimal speed values for each integral solution can be found using line search as discussed previously.

4.7 COMPUTATIONAL EXPERIMENTS

4.7.1 Implementation and Setup

We use commercial integer programming solver CPLEX [74] to implement the OA algorithm using branch-and-cut (B&C). Root node formulation is SSCPF-OA. Each power function $\alpha_i s_i^{\beta}$ is approximated at the points $\left\{L, \frac{L}{2}, \frac{L}{2^2}, \frac{L}{2^3}, \ldots, \frac{L}{2^t}, 0\right\}$ where t is the smallest integer satisfying $2^t \geq L$. The reason to use such initial approximation of the functions is that good solutions are expected to distribute the load over several processors and thus small speed values are more likely to be used compared to large speed values. Hence, we are approximating functions more precisely at the speed levels that are most likely going be used by near-optimal solutions. For $L = 1000$, root node model includes approximation of each function at $(2 + 10)$ points as $2^9 < 1000 < 2^{10}$. The relaxation solved at each node provides a lower bound for the tree rooted at that node. We do not interfere with CPLEX until an integral solution is found. Once an integral solution is found along one of the

branches, CPLEX calls our *lazy constraint callback* (LCC) implementation before accepting the solution.

In the LCC routine, we first obtain the optimal speed levels for selected processors by solving (4.9)-(4.10) using bisection as a line search method. Then, we approximate energy consumption curve of each processor turned on at its optimal speed value obtained from the line search. Note that once a processor is turned off through branching along one of the branches (i.e., once branching sets $x_i = 0$), we do not need to approximate its power function again along that branch. The approximations are added to the SSCPF-OA locally, i.e., the cuts are effective only for the tree rooted at the current node where integral solution is encountered. Compared to adding cuts globally, this strategy helps in keeping the model size at a moderate level along all the branches of the tree. If CPLEX encounters the same integral solution again, it implies that the solution is feasible and thus provides a valid upper bound for the search tree.

As Theorem 6 implies, when CPLEX terminates, we have an optimal integral solution and objective function value. However, the values of continuous variables may be different from their optimal values. This is due to the fact that any fractional solution along the tangent lines active at an integral solution yields the same objective function value. To understand why this is the case, one needs to notice that all the tangent lines active for a given integral solution have the same slope λ, as (4.8) suggests when $\mu_i = 0$, and $\sum s_i = L$. Our line search method indeed sets $\mu_i = 0$ for all processors that are turned on. Yet, it is straightforward to obtain optimal speed values using (4.9)-(4.10) once the optimal integral solution is known.

The DP implementation is a simple forward recursion starting with $T[0, 0]$ where we keep track of $\sum_{j \in T} (1/\alpha_j)^{1/(\beta-1)}$ for each γ value that has been discovered so far. As previously discussed, this allow us to compare T and $\{i\} \cup T$ in constant time.

In our computational experiments, we use several different number of processors (n) together with several different levels of the parameters L, α, β, and γ. For each setting, 10 replications are obtained and the minimum, median, and maximum of each measure of interest is reported. A PC with 3.00 GHz CPU and 3 GB RAM is used for all computations.

4.7.2 Results and Discussion

The first set of results aims at comparing CPLEX and our OA implementation. CPLEX is only able to solve quadratic programs and thus Table 11 summarizes results for instances where $\beta = 2$. We uniformly generated γ values between 1 000 and 10 000. Three different number of processors is used in experiments: $n = 25$, $n = 100$, and $n = 500$. Three levels of service load is used for each value of n: $L = 100$, $L = 500$, and $L = 1000$. For each combination of n and L, α values are generated uniformly between 100 and 500 or between 500 and 1000. One hour time limit and 0.1% relative termination gap is enforced for all experiments.

For $n = 25$, there are only 25 binary variables and both CPLEX and OA are able to solve instances to optimality very fast, though CPLEX sometimes needs almost 50 seconds for the $(L = 100, \alpha \sim U[100, 500])$ setting. For $n = 100$, CPLEX runs out of memory in less than half hour except for the $(L = 1000, \alpha \sim U[500, 1000])$ setting. The relative gap at termination is about 50% for $L = 100$, 20% for $L = 500$, and 5% for $L = 1000$. In comparison, OA implementation is able to obtain optimal solutions in less than 1 second for all settings. For $n = 500$, CPLEX runs out of memory in less than half hour with a relative termination gap above 55% for all instances. OA algorithm solves all problems to optimality in less than 20 seconds.

In Table 12, we summarize results for OA implementation where β values vary between 2 and 12. Though we use both $\beta \sim U[2, 6]$ and $\beta \sim U[6, 12]$ for $L = 100$; β values greater than 8 results in very large energy consumption values for $L = 500$ and $L = 1000$ (e.g., $(10^3)^{12}$), which CPLEX is not able to handle correctly due to numerical instabilities. Thus, we use only $\beta \sim U[2, 8]$ for $L > 100$. Table 12 also includes results for $n = 2500$ processors. The levels of other parameters used are similar to what we have used in Table 11. We also report the number of calls to the LCC routine and total number of cuts added during these calls.

For $n = 25$, 100, and 500, it is straight forward for OA to find optimal solutions. Less than 40 calls are made to the LCC through which at most 6000 cuts are added during B&C. When $n = 2500$, OA is still performing well in general, however, for some parameter settings

Table 11: Results for CPLEX vs. OA implementation.

$\beta = 2$, $\gamma \sim U[1\,000, 10\,000]$				**CPLEX**						**OA**					
				gap (%)			**time (sec)**			**gap (%)**			**time (sec)**		
n	L	α: low	α: high	min	med	max	min	med	max	min	med	max	min	med	max
25	100	100	500	0.00	0.00	0.00	5	20	48	0.01	0.07	0.10	0	0	0
		500	1 000	0.00	0.00	0.00	1	2	5	0.00	0.04	0.10	0	0	0
	500	100	500	0.00	0.00	0.00	0	0	0	0.00	0.00	0.10	0	0	0
		500	1 000	0.00	0.00	0.00	0	0	0	0.00	0.01	0.02	0	0	0
	1 000	100	500	0.00	0.00	0.00	0	0	0	0.00	0.00	0.01	0	0	0
		500	1 000	0.00	0.00	0.00	0	0	0	0.00	0.00	0.00	0	0	0
100	100	100	500	50.74	51.98	53.53	1 329	1 365	1 423†	0.06	0.08	0.10	0	0	1
		500	1 000	53.08	54.03	56.95	1 308	1 372	1 415†	0.01	0.06	0.08	0	0	1
	500	100	500	27.11	27.73	30.08	1 413	1 527	1 554†	0.03	0.06	0.10	0	0	1
		500	1 000	11.04	12.15	14.38	1 325	1 369	1 426†	0.00	0.01	0.02	0	0	0
	1 000	100	500	6.23	6.99	8.48	1 369	1 415	1 500†	0.00	0.00	0.00	0	0	0
		500	1 000	0.10	0.10	0.10	51	190	349	0.00	0.00	0.00	0	0	0
500	100	100	500	81.50	83.26	85.73	1 512	1 587	1 644†	0.02	0.08	0.10	5	6	19
		500	1 000	84.70	87.07	87.85	1 525	1 572	1 625†	0.04	0.08	0.10	3	4	5
	500	100	500	79.05	79.81	80.27	1 518	1 546	1 618†	0.04	0.09	0.10	4	4	4
		500	1 000	70.90	72.20	72.87	1 452	1 488	1 511†	0.04	0.09	0.10	3	4	4
	1 000	100	500	67.51	68.62	70.96	1 567	1 611	1 667†	0.06	0.09	0.10	3	3	4
		500	1 000	54.90	56.57	57.27	1 787	1 820	1 886†	0.02	0.09	0.10	3	3	4

† Out of memory.

it is not able to close optimality gap in one hour. Termination gap is less than 9% in the worst case, which requires almost half million cuts.

Table 13 summarizes the results comparing OA and DP. As expected, when the number of processors, n, gets large, branching based OA becomes overwhelmed by the size of the search space. When $n = 2500$, OA is able to close optimality gap only for the cases where it becomes easy to separate "*good*" and "*bad*" power functions from each other. DP is rather stable and its performance steadily degrades as the range of γ values gets larger. DP is powerful as it is able find the optimal solution in less than 5 minutes for all the instances we consider. The results suggest that OA is more suitable for small n whereas DP is more effective for small γ range.

4.8 ACKNOWLEDGEMENT

We wish to thank Dr. Kirk Pruhs, Dr. Daniel Mossé, and Daniel Cole in the Computer Science Department at University of Pittsburgh, for bringing this problem to our attention and valuable discussions.

Table 12: Results for OA implementation.

$\gamma \sim U[1\,000, 10\,000]$						**gap** (%)			**time** (sec)			**calls**			**cuts**		
n	L	α: low	α: high	β: low	β: high	min	med	max	min	med	max	min	med	max	min	med	max
25	100	100	500	2	6	0.00	0.06	0.10	0	0	0	2	7	12	50	140	246
				6	12	0.00	0.00	0.00	0	0	0	2	2	2	50	50	50
		500	1 000	2	6	0.00	0.00	0.00	0	0	0	2	2	2	50	50	50
				6	12	0.00	0.00	0.00	0	0	0	2	2	2	50	50	50
	500	100	500	2	8	0.00	0.01	0.10	0	0	0	2	2	2	50	50	50
		500	1 000	2	8	0.00	0.00	0.00	0	0	0	2	2	2	50	50	50
	1 000	100	500	2	8	0.00	0.00	0.00	0	0	0	2	2	2	50	50	50
		500	1 000	2	8	0.00	0.00	0.00	0	0	0	2	2	2	50	50	50
100	100	100	500	2	6	0.02	0.07	0.09	0	0	1	15	18	22	558	716	795
				6	12	0.02	0.07	0.10	0	1	1	10	14	18	867	1 223	1 575
		500	1 000	2	6	0.01	0.06	0.09	0	0	1	13	15	21	697	845	1 253
				6	12	0.05	0.08	0.10	1	1	1	10	13	15	925	1 197	1 375
	500	100	500	2	8	0.00	0.00	0.00	0	0	0	2	2	2	200	200	200
		500	1 000	2	8	0.00	0.00	0.00	0	0	0	2	2	2	200	200	200
	1 000	100	500	2	8	0.00	0.02	0.10	0	0	0	2	2	2	200	200	200
		500	1 000	2	8	0.00	0.01	0.05	0	0	0	2	2	2	200	200	200
500	100	100	500	2	6	0.04	0.08	0.10	4	4	7	28	37	69	1 850	2 093	3 078
				6	12	0.00	0.00	0.09	0	1	6	4	4	30	502	523	3 165
		500	1 000	2	6	0.02	0.06	0.10	4	5	10	33	38	100	3 423	3 749	7 820
				6	12	0.00	0.00	0.00	1	1	1	4	4	7	554	565	915
	500	100	500	2	8	0.03	0.08	0.10	3	3	3	12	16	20	3 698	4 494	5 406
		500	1 000	2	8	0.01	0.06	0.10	3	3	6	12	16	21	4 338	5 444	7 218
	1 000	100	500	2	8	0.00	0.07	0.09	1	2	2	4	7	10	1 672	2 911	3 912
		500	1 000	2	8	0.00	0.07	0.09	1	1	2	4	7	12	1 966	3 371	5 660
2 500	100	100	500	2	6	0.10	1.06	3.04	1 509	3 600	3 600	785	3 956	8 618	22 677	132 491	307 829
				6	12	0.00	0.01	0.03	3	3	4	4	4	4	576	609	636
		500	1 000	2	6	5.15	7.34	8.98	3 600	3 600	3 600	385	663	1 240	37 371	60 147	118 599
				6	12	0.00	0.00	0.08	3	3	3	4	4	4	666	688	694
	500	100	500	2	8	0.00	0.00	4.58	4	7	3 600	2	3	1 625	2 680	3 472	355 850
		500	1 000	2	8	0.00	0.00	0.00	4	8	9	2	4	4	2 712	4 141	4 212
	1 000	100	500	2	8	0.00	0.00	4.91	6	9	3 600	2	4	1 152	2 710	4 467	440 899
		500	1 000	2	8	0.00	0.00	2.22	4	10	3 600	2	4	762	2 708	4 377	523 411

Table 13: Results comparing OA and DP.

$\alpha \sim U[1,\, 1\,000]$				**OA**						**DP**		
$\gamma \sim U[1, \overline{\gamma}]$				**gap** (%)			**time** (sec)			**time** (sec)		
n	L	$\overline{\gamma}$	β	min	med	max	min	med	max	min	med	max
			3	0.08	0.62	0.97	1	1	1	0	1	1
		100	5	0.09	0.40	0.94	2	3	3	0	0	0
			8	0.18	0.41	0.96	3	3	4	0	0	0
			3	0.16	0.43	0.97	0	0	0	3	3	3
	50	1 000	5	0.04	0.47	0.86	1	1	2	2	2	3
			8	0.05	0.35	0.99	2	3	3	1	1	2
			3	0.24	0.72	0.86	0	0	0	8	11	15
		10 000	5	0.11	0.75	0.99	0	1	1	12	13	16
			8	0.01	0.62	0.90	2	2	2	10	12	15
			3	0.27	0.61	0.92	0	0	1	2	2	2
		100	5	0.57	0.77	0.97	1	2	2	1	1	1
			8	0.11	0.30	0.87	2	3	4	1	1	1
			3	0.22	0.76	0.99	0	0	0	6	8	9
500	100	1 000	5	0.31	0.71	1.00	1	1	1	7	8	9
			8	0.35	0.88	0.98	1	2	2	5	6	7
			3	0.47	0.78	0.94	0	0	3	27	34	40
		10 000	5	0.08	0.67	0.87	0	0	0	38	41	45
			8	0.18	0.72	0.96	1	1	1	37	40	52
			3	0.15	0.49	0.96	0	0	1	3	3	3
		100	5	0.36	0.62	0.89	1	1	1	3	3	3
			8	0.60	0.75	0.99	2	2	2	2	3	3
			3	0.14	0.49	0.84	0	0	1	24	26	29
	250	1 000	5	0.24	0.79	0.89	1	1	1	25	26	32
			8	0.30	0.80	1.00	2	2	2	22	23	28
			3	0.23	0.71	0.98	0	0	0	100	114	129
		10 000	5	0.36	0.81	0.99	0	1	1	156	168	182
			8	0.24	0.90	0.99	1	1	2	167	179	204

Table 13: (continued).

$\alpha \sim U[1, 1\,000]$ $\gamma \sim U[1, \overline{\gamma}]$				**OA**						**DP**		
				gap (%)			**time** (sec)			**time** (sec)		
n	L	$\overline{\gamma}$	β	min	med	max	min	med	max	min	med	max
			3	65.95	70.52	80.41	3 600	3 600	3 600	4	4	5
		100	5	74.03	79.30	83.66	3 600	3 600	3 600	3	3	3
			8	72.05	80.80	82.91	3 600	3 600	3 600	2	2	3
			3	0.87	33.81	49.13	6	3 600	3 600	9	10	11
	50	1 000	5	48.63	50.89	71.15	3 600	3 600	3 600	5	7	7
			8	52.70	63.51	78.65	3 598	3 600	3 600	4	5	5
			3	0.05	0.73	0.99	5	5	6	16	26	37
		10 000	5	0.22	27.09	59.05	9	3 600	3 600	20	24	33
			8	31.98	54.76	59.14	3 600	3 600	3 600	14	17	25
			3	47.98	56.80	73.64	3 600	3 600	3 600	8	9	9
		100	5	75.65	79.54	83.64	3 600	3 600	3 600	5	5	6
			8	71.33	77.43	80.20	3 600	3 600	3 600	4	4	6
			3	0.03	0.97	23.46	4	5	3 600	22	23	25
2 000	100	1 000	5	42.95	46.72	53.57	3 600	3 600	3 600	16	18	21
			8	52.88	57.86	63.25	3 600	3 600	3 600	11	13	15
			3	0.02	0.54	0.98	3	3	5	63	73	83
		10 000	5	0.41	0.89	0.99	4	4	7	69	80	86
			8	23.88	28.65	35.36	3 598	3 600	3 600	61	74	87
			3	61.35	68.50	75.39	3 600	3 600	3 600	23	24	27
		100	5	79.45	84.18	87.30	3 600	3 600	3 600	14	16	17
			8	82.16	84.22	85.08	3 600	3 600	3 600	10	11	12
			3	0.13	4.50	32.62	4	1 802	3 600	75	84	91
	250	1 000	5	49.42	58.81	60.83	3 600	3 600	3 600	67	72	79
			8	69.56	72.90	75.08	3 598	3 600	3 600	52	55	61
			3	0.05	0.82	0.93	3	3	5	262	328	363
		10 000	5	0.55	11.54	21.95	5	3 600	3 600	339	371	907
			8	45.03	46.43	47.40	3 600	3 600	3 600	326	397	486

5.0 CONCLUSION

This dissertation covers three important problems motivated by military and power management applications. We study each problem to discover its underlying structure and formulate nonlinear / stochastic mathematical models that capture details specific to the problem's context. Using advanced optimization techniques, we develop specialized exact, approximate, and heuristic solution methods that are directly applicable in real-life scenarios. The range of optimization techniques employed include decompositions, reformulations, and linearizations. Proposed approaches are evaluated and validated through extensive computational experiments and simulations.

In Chapter 2, we introduce the phased array antenna design problem to improve the imaging capability of the current antenna technology. Given that the problem is new to the Operations Research literature, we provide a complete treatment of the subject. A formal definition of the problem is given and a nonlinear mathematical model of the problem based on a novel use of information-theoretic entropy concept is formulated. This model is subsequently reformulated and linearized without sacrificing its strength. Next, an advanced decomposition method is proposed and implemented. To solve larger size instances, heuristic/approximation algorithms that synthesize old and new ideas are developed. Computational experiments show that our algorithms can handle large-scale instances efficiently. Phased array antenna simulations demonstrate effectiveness of our designs in improving antenna images.

Our work on phased array antenna is based on an empirical observation regarding the relationship between subarray geometry and antenna beam formation. Antenna is thus a black box in our implementation. However, antenna beam formation follows well-defined physical concepts and there is room for improvement through use of these concepts. As a

future research direction, development of models incorporating physics of beam formation into the proposed entropy concept might provide superior designs.

A two stage stochastic extension of the deterministic linear assignment problem (LAP) is studied in Chapter 3. The well-known LAP has many applications, including weapon-target assignment. We characterize a necessary optimality condition and use it in conjunction with the well-known Hungarian Method to develop a superior approximation algorithm. The proposed algorithm preserves the best known approximation bound and is shown to perform better than previously proposed algorithms on several instances with special structure.

The problem setting studied in this dissertation can hardly be the only interesting extension of LAP, which has numerous variants. Interesting future research directions include study of stochastic extensions with different objective functions, the multi-stage setting, extensions where availability of agents / jobs across stages is subject to change, and algorithms with better approximation guarantees or inapproximability results.

Finally, we discuss an actively studied problem from Computer Science literature in Chapter 4. Following the technological shift in the last decade, data warehouses with thousands of processors are becoming one of the major budget items for companies and institutions with large transaction volumes. In this environment, electricity becomes the major cost item and optimal processor scheduling that minimizes the total energy consumption is a crucial issue. We formulate a mathematical model of the problem, identify polynomially solvable instances, and provide a dynamic programming approach which we convert into a FPTAS. A successful implementation of the outer approximation algorithm is also presented. All the solution approaches are tested through computational experiments.

The setting of speed scaling problem considered in this dissertation is a static one. We do not consider the variation of service load over time and the policies for such dynamic environment. Yet the static case proves to be a hard one. As a future research direction, one might study the setting where service load is given as a set of jobs that need to be scheduled over time. Such an extension will require the use of scheduling tools together with different performance metrics. The online version is also interesting due to the nature of the problem tackled. Energy consumption and heat dissipation are closely related and their joint study is yet another promising research direction.

BIBLIOGRAPHY

[1] S. Ahmed, M. Tawarmalani, and N. Sahinidis, *A finite branch and bound algorithm for two-stage stochastic integer programs*, Mathematical Programming, 100 (2004), pp. 355–377.

[2] M. Albareda-Sambola, M. van der Vlerk, and E. Fernandez, *Exact solutions to a class of stochastic generalized assignment problems*, European Journal of Operational Research, 173 (2006), pp. 465–487.

[3] S. Albers, *Algorithms for Dynamic Speed Scaling*, in 28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011), T. Schwentick and C. Dürr, eds., vol. 9 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2011, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 1–11.

[4] S. Albers, A. Antoniadis, and G. Greiner, *On multi-processor speed scaling with migration: extended abstract*, in Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures, SPAA '11, New York, NY, USA, 2011, ACM, pp. 279–288.

[5] S. Albers and H. Fujiwara, *Energy-efficient algorithms for flow time minimization*, ACM Transactions on Algorithms, 3 (2007).

[6] S. Albers, F. Müller, and S. Schmelzer, *Speed scaling on parallel processors*, in Proceedings of the 19th annual ACM symposium on Parallel algorithms and architectures, SPAA '07, New York, NY, USA, 2007, ACM, pp. 289–298.

[7] A. Alonso-Ayuso, L. Escudero, and M. Teresa Ortuño, *BFC, a branch-and-fix coordination algorithmic framework for solving some types of stochastic pure and mixed 0–1 programs*, European Journal of Operational Research, 151 (2003), pp. 503–519.

[8] J. Ash and S. Golomb, *Tiling deficient rectangles with trominoes*, Mathematics Magazine, 77 (2004), pp. 46–55.

[9] A. Atamtürk, G. Nemhauser, and M. Savelsbergh, *A combined lagrangian, linear programming, and implication heuristic for large-scale set partitioning problems*, Journal of Heuristics, 1 (1996), pp. 247–259.

[10] P. Avella, A. Sassano, and I. Vasil'ev, *Computational study of large-scale p-median problems*, Mathematical Programming, 109 (2007), pp. 89–114.

[11] A. Bains and T. Biedl, *Reconstructing hv-convex multi-coloured polyominoes*, Theoretical Computer Science, 411 (2010), pp. 3123–3128.

[12] E. Balas and M. Padberg, *On the set-covering problem*, Operations Research, 20 (1972), pp. pp. 1152–1161.

[13] ——, *Set partitioning: A survey*, SIAM Review, 18 (1976), pp. pp. 710–760.

[14] N. Bansal, H.-L. Chan, T.-W. Lam, and L.-K. Lee, *Scheduling for speed bounded processors*, in Automata, Languages and Programming, L. Aceto, I. Damgård, L. Goldberg, M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, eds., vol. 5125 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2008, pp. 409–420.

[15] N. Bansal, H.-L. Chan, and K. Pruhs, *Speed scaling with an arbitrary power function*, in Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '09, Philadelphia, PA, USA, 2009, Society for Industrial and Applied Mathematics, pp. 693–701.

[16] N. Bansal, T. Kimbrel, and K. Pruhs, *Speed scaling to manage energy and temperature*, Journal of the ACM, 54 (2007), pp. 1–39.

[17] N. Bansal, K. Pruhs, and C. Stein, *Speed scaling for weighted flow time*, in Proceedings of the 18th annual ACM-SIAM symposium on Discrete algorithms, SODA '07, Philadelphia, PA, USA, 2007, Society for Industrial and Applied Mathematics, pp. 805–813.

[18] G. Barequet, M. Moffie, A. Ribó, and G. Rote, *Counting polyominoes on twisted cylinders*, in Proceedings of 2005 European Conference on Combinatorics, Graph Theory and Applications (EuroComb '05), Ed. S. Felsner, Discrete Mathematics and Theoretical Computer Science Proceedings AE, 2005, pp. 369–374.

[19] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance, *Branch-and-price: Column generation for solving huge integer programs*, Operations Research, 46 (1998), pp. 316–329.

[20] L. A. Barroso, *The price of performance*, Queue, 3 (2005), pp. 48–53.

[21] C. Berge, C. Chen, V. Chvatal, and C. Seow, *Combinatorial properties of polyominoes*, Combinatorica, 1 (1981), pp. 217–224.

[22] R. Berger, *The undecidability of the domino problem*, Mem. Amer. Math. Soc. No., 66 (1966), p. 72.

[23] B. Bingham and M. Greenstreet, *Energy optimal scheduling on multiprocessors with migration*, in International Symposium on Parallel and Distributed Processing with Applications, 2008, ISPA '08, 2008, pp. 153 –161.

[24] J. Birge and F. Louveaux, *Introduction to Stochastic Programming*, Springer, 1997.

[25] O. Bodini, *Tiling a rectangle with polyominoes*, Discrete Mathematics and Theoretical Computer Science, (2003), pp. 81–88.

[26] P. Bonami, L. T. Biegler, A. R. Conna, G. Cornuejols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wachter, *An algorithmic framework for convex mixed integer nonlinear programs*, Discrete Optimization, 5 (2008), pp. 186–204.

[27] P. Bonami, M. Kilinç, and J. Linderoth, *Algorithms and software for convex mixed integer nonlinear programs*, in Mixed Integer Nonlinear Programming, J. Lee and S. Leyffer, eds., vol. 154 of The IMA Volumes in Mathematics and its Applications, Springer New York, 2012, pp. 1–39.

[28] F. Bower, D. Sorin, and L. Cox, *The impact of dynamically heterogeneous multicore processors on thread scheduling*, Micro, IEEE, 28 (2008), pp. 17 –25.

[29] P. Brass, W. Moser, and J. Pach, *Research Problems in Discrete Geometry*, Springer, New York, 2005.

[30] K. M. Bretthauer and B. Shetty, *The nonlinear resource allocation problem*, Operations Research, 43 (1995), pp. pp. 670–683.

[31] K. M. Bretthauer and B. Shetty, *The nonlinear knapsack problem – algorithms and applications*, European Journal of Operational Research, 138 (2002), pp. 459 – 472.

[32] E. Brookner, *Phased arrays for the new millennium*, in Phased Array Systems and Technology, 2000. Proceedings. 2000 IEEE International Conference on, 2000, pp. 3 –19.

[33] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment problems*, Cambridge University Press, 2012.

[34] H. Cai, J. Liu, Y. Chen, and H. Wang, *Survey of the research on dynamic weapon-target assignment problem*, Journal of Systems Engineering and Electronics, 17 (2006), pp. 559 – 565.

[35] J. Cardinal, S. Fiorini, and G. Joret, *Tight results on minimum entropy set cover*, Algorithmica, 51 (2008), pp. 49–60.

[36] G. Castiglione, A. Frosini, E. Munarini, A. Restivo, and S. Rinaldi, *Combinatorial aspects of l-convex polyominoes*, European Journal of Combinatorics, 28 (2007), pp. 1724–1741.

[37] COIN-OR, *Branch-Cut-Price Framework 1.3.1*, https://projects.coin-or.org/Bcp, 2011.

[38] D. Cole, S. Im, B. Moseley, and K. Pruhs, *Speed scaling for stretch plus energy*, Operations Research Letters, 40 (2012), pp. 180 – 184.

[39] C. D'Ambrosio and S. Martello, *Heuristic algorithms for the general nonlinear separable knapsack problem*, Computers & Operations Research, 38 (2011), pp. 505 – 513.

[40] M. Dehmer and F. Emmert-Streib, *Structural information content of networks: Graph entropy based on local vertex functionals*, Computational Biology and Chemistry, 32 (2008), pp. 131–138.

[41] M. Dehmer and A. Mowshowitz, *A history of graph entropy measures*, Information Sciences, 181 (2011), pp. 57 – 78.

[42] M. Delest and G. Viennot, *Algebraic languages and polyominoes enumeration*, Theoretical Computer Science, 34 (1984), pp. 169–206.

[43] E. Demaine and M. Demaine, *Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity*, Graphs and Combinatorics, 23 (2007), pp. 195–208.

[44] J. Dennis and R. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Classics in Applied Mathematics, Society for Industrial and Applied Mathematics, 1987.

[45] K. Dhamdhere, V. Goyal, R. Ravi, and M. Singh, *How to pay, come what may: Approximation algorithms for demand-robust covering problems*, in Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on, 2005, pp. 367–378.

[46] K. Dhamdhere, R. Ravi, and M. Singh, *On stochastic minimum spanning trees*, in Proceedings of the 11th International Conference on Integer Programming and Combinatorial Optimization, 2005.

[47] E. Duchi, S. Rinaldi, and G. Schaeffer, *The number of z-convex polyominoes*, Advances in Applied Mathematics, 40 (2008), pp. 54–72.

[48] M. A. Duran and I. E. Grossmann, *An outer-approximation algorithm for a class of mixed-integer nonlinear programs*, Mathematical Programming, 36 (1986), pp. 307–339.

[49] M. Dyer and L. Stougie, *Computational complexity of stochastic programming problems*, Mathematical Programming, 106 (2006), pp. 423–432.

[50] B. Escoffier, L. Gourves, J. Monnot, and O. Spanjaard, *Two-stage stochastic matching and spanning tree problems: Polynomial instances and approximation*, European Journal of Operations Research, 205 (2010), pp. 19–30.

[51] S. Fang, J. Rajasekera, and H. Tsao, *Entropy optimization and mathematical programming*, International series in operations research & management science, Kluwer Academic Publishers, 1997.

[52] A. Flaxman, A. Frieze, and M. Krivelevich, *On the random 2-stage minimum spanning tree*, Random Structures and Algorithms, 28 (2006), pp. 24–36.

[53] R. Fletcher and S. Leyffer, *Solving mixed-integer nonlinear programs by outer approximation*, Mathematical Programming, 66 (1994), pp. 327–349.

[54] H. Fukuda, N. Mutoh, G. Nakamura, and D. Schattschneider, *A method to generate polyominoes and polyiamonds for tilings with rotational symmetry*, Graphs and Combinatorics, 23 (2007), pp. 259–267.

[55] M. R. Garey and D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, Series of books in the mathematical sciences, W. H. Freeman, 1979, ch. Appendix: A list of NP-Complete Problems, p. 257.

[56] ——, *Computers and intractability: a guide to the theory of NP-completeness*, Series of books in the mathematical sciences, W. H. Freeman, 1979.

[57] R. Garfinkel and G. Nemhauser, *The set-partitioning problem: Set covering with equality constraints*, Operations Research, 17 (1969), pp. pp. 848–856.

[58] A. Ghoniem and H. Sherali, *Complementary column generation and bounding approaches for set partitioning formulations*, Optimization Letters, 3 (2009), pp. 123–136.

[59] S. Golomb, *Tiling with polyominoes*, Journal of Combinatorial Theory, 1 (1966), pp. 280–296.

[60] ——, *Tiling with sets of polyominoes*, Journal of Combinatorial Theory, 9 (1970), pp. 60–71.

[61] ——, *Polyominoes: Puzzles, Patterns, Problems, and Packings*, Princeton University Press, 2nd ed., 1994.

[62] D. Golovin, V. Goyal, and R. Ravi, *Pay today for a rainy day: improved approximation algorithms for demand-robust min-cut and shortest path problems*, in STACS 2006, Springer, 2006, pp. 206–217.

[63] O. Günlük, J. Lee, and R. Weismantel, *MINLP strengthening for separable convex quadratic transportation-cost UFL*, tech. rep., Technical Report RC24213 (W0703-042), IBM Research Division, 2007.

[64] A. Gupta, S. Im, R. Krishnaswamy, B. Moseley, and K. Pruhs, *Scheduling heterogeneous processors isn't as easy as you think*, in Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '12, SIAM, 2012, pp. 1242–1253.

[65] A. Gupta, R. Krishnaswamy, and K. Pruhs, *Nonclairvoyantly scheduling power-heterogeneous processors*, Sustainable Computing: Informatics and Systems, 1 (2011), pp. 248 – 255.

[66] A. Gupta, R. Ravi, and A. Sinha, *LP rounding approximation algorithms for stochastic network design*, Mathematics of Operations Research, 32 (2007), pp. 345–364.

[67] E. Halperin and R. Karp, *The minimum-entropy set cover problem*, Theoretical Computer Science, 348 (2005), pp. 240 – 250.

[68] X. Han, T.-W. Lam, L.-K. Lee, I. K. To, and P. W. Wong, *Deadline scheduling and power management for speed bounded processors*, Theoretical Computer Science, 411 (2010), pp. 3587 – 3600.

[69] G. Hardy, J. Littlewood, and G. Pólya, *Inequalities*, Cambridge Mathematical Library, Cambridge University Press, 1952.

[70] U. Haus, D. Michaels, and A. Savchenko, *Extended formulations for MINLP problems by value decompositions*, in EngOpt 2008 – International Conference on Engineering Optimization, 2008.

[71] D. S. Hochbaum, *Lower and upper bounds for the allocation problem and other nonlinear optimization problems*, Mathematics of Operations Research, 19 (1994), pp. pp. 390–409.

[72] D. S. Hochbaum, *A nonlinear knapsack problem*, Operations Research Letters, 17 (1995), pp. 103 – 110.

[73] T. Ibaraki and N. Katoh, *Resource allocation problems: algorithmic approaches*, MIT Press series in the foundations of computing, MIT Press, 1988.

[74] IBM ILOG CPLEX 12, *http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/*.

[75] S. Irani, S. Shukla, and R. Gupta, *Algorithms for power savings*, in Proceedings of the 14th annual ACM-SIAM symposium on Discrete algorithms, SODA '03, Philadelphia, PA, USA, 2003, Society for Industrial and Applied Mathematics, pp. 37–46.

[76] I. Jensen, *Enumerations of lattice animals and trees*, Journal of Statistical Physics, 102 (2001), pp. 865–881.

[77] J. Kapur and H. Kesavan, *Entropy optimization principles with applications*, Academic Press, 1992.

[78] S. Karademir, N. Kong, and O. A. Prokopyev, *On greedy approximation algorithms for a class of two-stage stochastic assignment problems*, Optimization Methods and Software, ahead-of-print (2012), pp. 1–26.

[79] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack problems*, Springer Verlag, 2004.

[80] C. Ko, J. Lee, and M. Queyranne, *An exact algorithm for maximum entropy sampling*, Operations Research, 43 (1995), pp. pp. 684–691.

[81] N. Kong and A. Schaefer, *A factor 1/2 approximation algorithm for two-stage stochastic matching problems*, European Journal of Operational Research, 81 (2006), pp. 387–394.

[82] M. Köppe, Q. Louveaux, and R. Weismantel, *Intermediate integer programming representations using value disjunctions*, Discrete Optimization, 5 (2008), pp. 293–313.

[83] H. Kuhn, *The hungarian method for the assignment problem*, Naval Research Logistics Quarterly, 2 (1955), pp. 83–97.

[84] T.-W. Lam, L.-K. Lee, H.-F. Ting, I. To, and P. Wong, *Sleep with guilt and work faster to minimize flow plus energy*, in Automata, Languages and Programming, S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. Nikoletseas, and W. Thomas, eds., vol. 5555 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2009, pp. 665–676.

[85] T.-W. Lam, L.-K. Lee, I. To, and P. Wong, *Speed scaling functions for flow time scheduling based on active job count*, in Algorithms - ESA 2008, D. Halperin and K. Mehlhorn, eds., vol. 5193 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2008, pp. 647–659.

[86] T.-W. Lam, L.-K. Lee, I. K. K. To, and P. W. H. Wong, *Competitive non-migratory scheduling for flow time and energy*, in Proceedings of the 20th annual symposium on Parallelism in algorithms and architectures, SPAA '08, New York, NY, USA, 2008, ACM, pp. 256–264.

[87] J. Lee, *Constrained maximum-entropy sampling*, Operations Research, 46 (1998), pp. pp. 655–664.

[88] R. Mailloux, *Phased array theory and technology*, Proceedings of the IEEE, 70 (1982), pp. 246–302.

[89] R. Mailloux, S. Santarelli, and T. Roberts, *Wideband arrays using irregular (polyomino) shaped subarrays*, Electronics Letters, 42 (2006), pp. 11–12.

[90] R. Mailloux, S. Santarelli, T. Roberts, and D. Luu, *Irregular polyomino-shaped subarrays for space-based active arrays*, International Journal of Antennas and Propagation, 2009 (2009). Article ID 956524, 9 pages.

[91] W. Marshall, *Packing rectangles with congruent polyominoes*, Journal of Combinatorial Theory, 77 (1997), pp. 181–192.

[92] R. Mollin, *An introduction to cryptography*, Discrete mathematics and its applications, Chapman & Hall/CRC, 2007.

[93] C. Moore and J. Robson, *Hard tiling problems with simple tiles*, Discrete & Computational Geometry, 26 (2001), pp. 573–590.

[94] R. A. Murphey, *Integrated assignment and path planning*, PhD thesis, University of Florida, 2005.

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

[96] L. Ntaimo and S. Sen, *A comparative study of decomposition algorithms for stochastic combinatorial optimization*, Computational Optimization and Applications, 40 (2008), pp. 299–319.

[97] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*, Dover books on mathematics, Dover Publications, 1998.

[98] D. Parker and D. Zimmermann, *Phased arrays - part I: Theory and architectures*, IEEE Transactions on Microwave Theory and Techniques, 50 (2002), pp. 678–687.

[99] E. L. Pasiliao, *Algorithms for multidimensional assignment problems*, PhD thesis, University of Florida, 2003.

[100] I. Quesada and I. E. Grossmann, *An LP/NLP based branch and bound algorithm for convex MINLP optimization problems*, Computers & Chemical Engineering, 16 (1992), pp. 937–947.

[101] M. Reid, *Tiling with similar polyominoes*, Journal of Recreational Mathematics, 31 (2002), pp. 15–24.

[102] A. Ruszczyński and A. Shapiro, eds., *Handbooks in OR&MS: Stochastic Programming, Vol. 10*, Elsevier, 2003.

[103] D. Ryan and B. Foster, *An integer programming approach to scheduling*, Computer scheduling of public transport urban passenger vehicle and crew scheduling, (1981), pp. 269–280.

[104] R. Schultz, *Stochastic programming with integer variables*, Mathematical Programming, 97 (2003), pp. 285–309.

[105] S. Sen and H. Sherali, *Decomposition with branch-and-cut approaches for two-stage stochastic mixed-integer programming*, Mathematical Programming, 106 (2006), pp. 203–223.

[106] C. Shannon, *A mathematical theory of communication*, Bell System Technical Journal, 27 (1948), pp. 379–423.

[107] A. Shapiro, D. Dentcheva, and A. Ruszczyński, *Lectures on Stochastic Programming: Modeling and Theory*, Society for Industrial and Applied Mathematics, 2009.

[108] H. Sherali and X. Zhu, *On solving discrete two-stage stochastic programs having mixed-integer first- and second-stage variables*, Mathematical programming, 108 (2006), pp. 597–616.

[109] G. Simonyi, *Graph entropy: A survey*, Combinatorial Optimization, 20 (1995), pp. 399–441.

[110] M. van der Vlerk, *Stochastic programming with simple integer recourse*, in Encyclopedia of Optimization, C. Floudas and P. Pardalos, eds., Springer, 2009, pp. 3795–3797.

[111] F. Vanderbeck, *On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm*, Operations Research, 48 (2000), pp. pp. 111–128.

[112] ——, *Branching in branch-and-price: a generic scheme*, Mathematical Programming, 130 (2011), pp. 249–294.

[113] V. V. Vazirani, *Approximation algorithms*, springer, 2004.

[114] P. Wang, *2 algorithms for constrained two-dimensional cutting stock problems*, Operations Research, 31 (1983), pp. 573–586.

[115] T. Westerlund and F. Pettersson, *An extended cutting plane method for solving convex MINLP problems*, Computers & Chemical Engineering, 19 (1995), pp. S131–S136.

[116] F. Yao, A. Demers, and S. Shenker, *A scheduling model for reduced CPU energy*, in Proceedings of 36th Annual Symposium on Foundations of Computer Science, 1995, pp. 374 –382.