12-2019

Optimizing Block-Stacking Operations with Relocation

Hueon Lee
University of Arkansas, Fayetteville

Optimizing Block-Stacking Operations with Relocation

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Engineering with a concentration in Industrial Engineering

by

Hueon Lee
Korea Aerospace University
Bachelor of Science in Logistics Management, 2007
Korea Aerospace University
Master of Science in Logistics, 2009

December 2019
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council.

_____
John Austin White, Jr., Ph.D.
Dissertation Co-Director

_____
Kelly Sullivan, Ph.D.
Dissertation Co-Director

_____
Shengfan Zhang
Dissertation Co-Director

_____
Sunderesh S. Heragu, Ph.D.
Ex-Officio Member

Abstract

The focus of the dissertation is developing the optimization problem of finding the minimum-cost operational plan of block stacking with relocation as well as devising a solution procedure to solve practical-sized instances of the problem. Assuming changeable row depth instead of permanent row depth, this research is distinguished from conventional block stacking studies.

The first contribution of the dissertation is the development of the optimization problem under the assumption of deterministic demand. The problem is modeled using integer programming as a variation of the unsplittable multi-commodity flow problem. To find a good feasible solution of practical-sized instances in reasonable time, we decompose the original problem into a series of generalized assignment problems. In addition, to establish a good lower bound on the optimal objective function value, we apply a relaxation based upon Lagrangean decomposition in which the relaxed problem separates into a set of shortest path problems and a set of binary knapsack problems.

The second contribution of the dissertation is the development of the optimization problem under the assumption of stochastic demand. The problem is formulated as a discrete time finite horizon Markov decision process model, incorporating the recursive daily situation of determining the assignment of product lots to storage areas for a day based on uncertain daily demand and observed system information. To tackle computational intractability in solving practical-sized instances, we develop a heuristic solution approach taking an on-line manner by instantly determining an action for a single observed state rather than an off-line manner by predetermining an action for every state.

Table of Contents

List of Tables

List of Figures

Selected Abbreviations

DBS	dynamic block stacking
SDBS	semi-dynamic block stacking
SBS	static block stacking
BSSPwRuDD	block stacking single product with relocation under deterministic demand
BSMPwRuDD	block stacking multiple product with relocation under deterministic demand
BSSPwRuSD	block stacking single product with relocation under stochastic demand
BSMPwRuSD	block stacking multiple product with relocation under stochastic demand
OC	daily operating cost of a single product lot
FS	floor space cost of a single product lot
ST	replenishment cost of a single product lot
RT	retrieval cost of a single product lot
BT	relocation cost of a single product lot
THD	time horizon decomposition
LD	Lagrangean decomposition
MDP	Markov decision process
IDH	Instant determination heuristic
GAP	generalized assignment problem
OFV	objective function value
PMF	probability math function

CHAPTER 1. Introduction

This dissertation investigates block-stacking operations. Specifically, optimization models are developed for block-stacking operations when the relocation of unit loads is allowed. Optimization models are mathematically formulated and solution procedures are developed to solve practical-sized instances of block-stacking operations. The results of the study can benefit a business operating a block stacking warehouse by reducing the combination of storage space and material handling costs.

1. Research Motivation

Block stacking is a storage method commonly used in a warehouse for unitized items. With limited or no supporting equipment, unit loads are stacked on top of each other and arranged in stacks within rows on the floor. Figure 1.1 illustrates the arrangement of unit loads in stacks and rows when block stacking is employed.

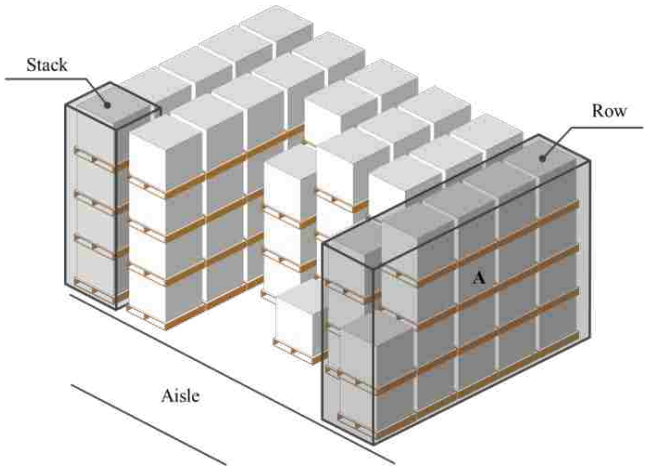

Figure 1.1: Example of block stacking

Compared to the storage of unit loads in selective storage rack, block stacking storage is characterized as having restricted accessibility to stored unit loads and having smaller equipment

investment cost. Weighing the pros and cons of accessibility and equipment cost, block stacking is preferred for the long-term storage of slow-turnover items and for the storage of large-volume and fast-moving lots (Ross, 1993); thus, block stacking is often used to store appliances, food and beverages, household product, tires, bags of potting mix and fertilizer, and construction materials, among other products (Matson and White, 1982, Sonnentag et al, 2014).

In planning block- stacking storage, choosing the row depth (maximum number of stacks in the row) for a product lot is important. With the stack height (maximum number of unit loads in the stack) generally given as a fixed value for a product lot, the row depth determines how unit loads are aligned on the floor over its storage life. The row depth dictates the extent of the space loss caused by last-in, first-out storage and retrieval in a storage row and affects material handling efficiency.

A fundamental assumption of previous block-stacking research is the row depth designated for a product lot is permanent and not changeable. Because changing row depth for a product during its storage life requires additional material handling and can cause difficulty in inventory management, changing row depth is rarely considered in practice. Therefore, previous research has tended to be based on an assumption of permanent row depth.

However, as demonstrated in this research, prohibiting changes in row depth over a product's storage life can lead to a requirement for significantly greater storage space than is required when row depths are allowed to change during a product's storage life. Likewise, when employing block stacking in an existing warehouse, by allowing row depths to be changed during a product's storage life, the storage capacity of the warehouse can be increased significantly. In the end, allowing storage depth to change is a decision involving tradeoffs of capital cost (storage space) and operating cost (material handling).

2. Research Organization

The remainder of the dissertation is organized as follows:

- CHAPTER 2. Block Stacking Multiple Products with Relocation under Deterministic Demand
- CHAPTER 3. Block Stacking Multiple Products with Relocation under Stochastic Demand
- CHAPTER 4. Conclusions and Future Research
- APPENDIX A. Cost Model of Dynamic Block Stacking Operations
- APPENDIX B. The Dynamic Block Stacking Problem with Random Demand

In CHAPTER 2, under the assumption of deterministic demand, the optimization problem of finding the minimum-cost operational plan of block stacking with relocation is formulated. The problem is modeled using integer programming as a variation of the unsplittable multi-commodity flow problem. To find a good feasible solution of practical-sized instances in a reasonable time, we decompose the original problem into a series of generalized assignment problems. In addition, to establish a good lower bound on the optimal objective function value, we apply a relaxation based upon Lagrangean decomposition in which the relaxed problem separates into a set of shortest path problems and a set of binary knapsack problems. CHAPTER 2 contributes (i) the first model for block stacking multiple products with changeable row depth, (ii) a solution method based on a strategy of decomposing the original problem into smaller and easier-to-solve sub-problems and, (iii) the quantitative analysis of the block stacking storage system's behavior when changing row depth by relocation is allowed.

In CHAPTER 3, under the assumption of stochastic demand, the optimization problem of determining the minimum-cost operational plan of block stacking with relocation is developed. The problem is formulated as a discrete time, finite horizon, discrete event Markov decision process model, incorporting the recursive daily situation of determining the assignment of

product lots to storage areas for a day based on uncertain daily demand and observed system information. To tackle computational intractability in solving practical-sized instances, we develop a heuristic solution approach by taking an on-line manner by instantly determining an action for a single observed state rather than an off-line manner by predetermining an action for every state. CHAPTER 3 contributes (i) the first model for block stacking multiple products with changeable row depth under stochastic demand and (ii) a solution method based on sampling technique guaranteeing reliability and efficiency in solving practical-sized instances.

CHAPTER 4 summarizes the dissertation and presents conclusions drawn from the findings of the study. It provides an overall insight into block stacking with changeable row depth and addresses how the research results can be used in operating a block stacking system. In addition, it includes recommendations for further research.

APPENDIX A introduces a block stacking cost model with the changeable row depth. The developed model differs from existing cost models by including relocation cost. It is based on a single product lot's daily operations and thus, easily modeled as the cost function of a daily operational plan of block stacking. In developing the optimization problems of CHAPTER 2 and CHAPTER 3, we assume a cost function derived from the cost model provided in APPENDIX A.

APPENDIX B contains a reprint of a published proceeding paper, "The Dynamic Block Stacking Problem with Random Demand"; it addresses block stacking a single product with relocation under stochastic demand. Several technical terms and notations in the original paper are revised in APPENDIX B to match terminology of the dissertation. As an extension of the earlier research, CHAPTER 3 is based on the mathematical model and the findings of APPENDIX B.

CHAPTER 2. Block Stacking Multiple Products with Relocation under Deterministic Demand

Abstract

An optimization model is formulated for block stacking multiple products with relocation under deterministic demand. It assumes changeable row depth instead of permanent row depth unlike the conventional optimization model for block stacking. It determines an assignment of product lots to storage areas each day given known inventory levels and daily demands over a time horizon to minimize total operating cost. The problem is modeled using integer programming as a variation of the unsplittable multi-commodity flow problem. To obtain good feasible solutions of practical-sized instances in reasonable time, we decompose the original problem into a series of generalized assignment problems. In addition, to establish a good lower bound on the optimal objective function value, we apply a relaxation based upon Lagrangean decomposition in which the relaxed problem separates into a set of shortest path problems and a set of binary knapsack problems. Comparing block stacking with changeable row depth and permanent row depth, the former requires less storage capacity and incurs less operating cost. The newly proposed block stacking uses floor space efficiently by timely changing row depth. It not only alleviates honeycomb loss and enables the product lot to yield occupied storage locations to another product lot if required. Its merit is magnified when storage capacity is insufficient based on the inventory level.

Keyword

Block stacking, deterministic demand, changeable row depth, multi-commodity flow problem, time horizon decomposition heuristic, Lagrangean decomposition heuristics

1. Introduction

Block-stacking storage, also known as floor storage, is a common storage method used in a warehouse. It consists of unitized items stacked on top of each other with stacks arranged on the floor; typically, limited or no supporting equipment is used. Compared to modern selective storage rack, block-stacking storage is old-fashioned and is somewhat inconvenient in placing and retrieving loads. It, however, can be an attractive alternative because of the low investment cost of equipment as long as it is applied in a suitable storage environment. See Tompkins et al. (2010) and Bartholdi and Hackman (2014) for brief descriptions of the block stacking storage method.

A block-stacking storage system is comprised of storage areas having the same or different depths, lengths, and heights. Unit loads are placed in these storage areas, forming stacks and rows in reserved positions. A stack is a vertical set of unit loads and a row is a horizontal set or line of stacks. Stacks are arrayed back-to-back in a row and rows are placed next to each other in the storage area. The stack positions and the row positions in the storage area are explicitly designated. We refer to the stack as a full stack if it fully occupies the volume of space reserved for the stack position; otherwise, it is referred to as a partial stack. Similarly, a full row and a partial row are defined. Figure 2.1 illustrates a block stacking storage system consisting of a 3-deep storage area and a 5-deep storage area. It includes full and partial stacks, full and partial rows, stack positions, and row positions.

When planning a block stacking storage system, a determination of how unit loads of each lot are to be arranged is an important issue. One can simply specify the stack height or the maximum number of unit loads in the stack and the row depth or maximum number of the stacks in the row. Consequently, one must determine the stack height and the row depth for each lot when designing

the system. In practice, the stack height is often determined by storage restrictions such as ceiling height, the storage/retrieval equipment used, and possible load crushing, whereas the row depth is adjustable. For this reason, finding an optimal row depth to minimize space cost and/or material handling cost has been the primary research topic associated with block stacking storage.

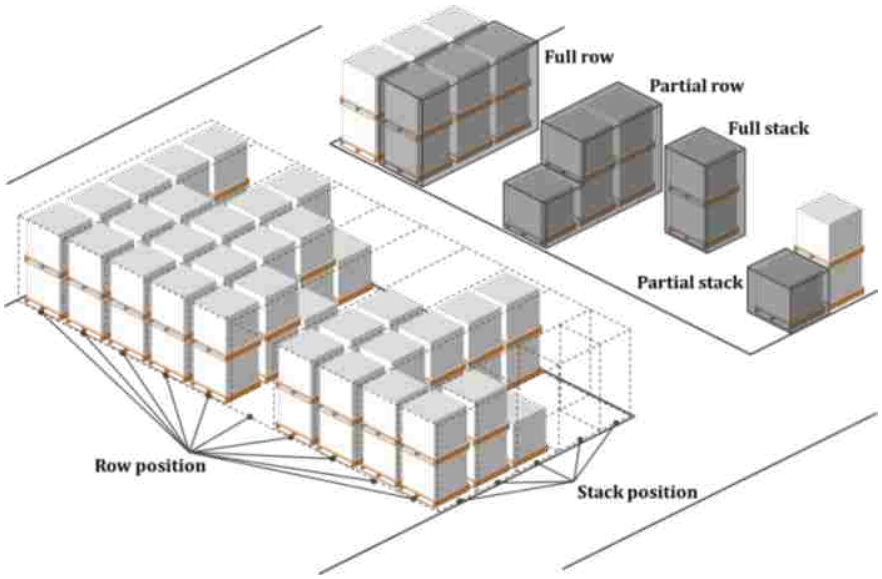

Figure 2.1: Instance of block stacking storage system

Row depth is closely related to the space utilization of the block stacking storage system. To ensure rotation of inventory and reduce the number of times a unit load is moved, generally, neither unit loads of different products nor unit loads of different lots of the same product are stored in the same row. (Bartholdi and Hackman, 2014). Following this storage rule, open space in the row position is not used until the position has been completely vacated. This unusable storage capacity is a peculiar phenomenon of the block stacking storage method; the incurred space loss is referred to as honeycombing (Tompkins et al., 2010). Accordingly, the row depth dictates the extent of the honeycomb loss and, resultantly, space utilization.

This operational issue of honeycomb loss led to previous studies focusing on finding an optimal row depth, one that minimizes operating cost incurred from required space and/or

material handling. To achieve the objective of optimizing space utilization and reducing material handling, researchers have tried to properly determine a single row depth for a single product lot, a set of row depths for a single product lot, a single row depth for multiple product lots, or a set of row depths for multiple product lots. When splitting a lot is allowed, a row depth is chosen for each separated portion of the lot in a given set of row depths. Considering its common employment in warehouses and its long history, a relatively small number of papers have been published on the subject.

A fundamental assumption of previous papers is a permanent row depth for a lot. In other words, researchers have assumed the row depth designated does not change over its storage life. Changing the row depth is possible by relocating remaining unit loads from the current designated storage area to a newly assigned storage area. The relocation, however, is rarely considered in practice because of additional material handling and the need to keep track of inventories. Thus, the permanent row depth is a very reasonable assumption. However, changing the row depth and relocating the remaining inventory can have considerable benefits by offsetting additional material handling with space savings. Specifically, changing the row depth can increase space utilization by reducing honeycomb loss and, consequently, decrease space cost.

The paper by Lee *et al*. (2016) appears to be the only published investigation of the impacts of allowing a lot's row depth to change by relocating its remaining inventory. They consider a single lot under stochastic demand conditions whereas we consider multiple lots and assume deterministic demand. Although we are not aware of other research that formally studies changing the row depth in a block stacking storage system, many papers have noted the possible benefits of relocation and recommended it as a topic for future research. (Kind, 1965, 1975; Roberts, 1968; Gavin, 1979; Goetschalckx and Ratliff, 1991; Ross, 1993). Kind (1965, 1975)

mentioned honeycomb losses can be reduced by relocating remaining unit loads in a mostly depleted lot to shorter rows, but recommended against constant re-warehousing. Ross (1993) noted relocation incurs considerable expense of labor and can result in damage to items. Roberts (1968) pointed out in some cases presently stored items in a storage row must be re-warehoused in order to assign adequate space to another product lot.

Unlike previous research, this paper relaxes the permanent row depth assumption in a block stacking system and allows changes in row depth by relocation. To distinguish block stacking with relocation and without relocation, we refer to the former as Dynamic Block Stacking (DBS) and the latter as Static Block Stacking (SBS). We address the dynamic block stacking optimization problem under deterministic demand and determine the row depths for product lots each day given known inventory levels and daily demands over a time horizon to minimize total operating cost. The row depth represents the storage area where a product lot is accommodated during the day. The problem is referred to as Block Stacking Multiple Product with Relocation under Deterministic Demand (BSMPwRuDD) and its solution defining which product lot is stored in which storage area for a day over a time horizon is called a dynamic block stacking plan or DBS plan. In developing the mathematical model of the problem, we construct a cost model based on daily operations such as storage, replenishment, retrieval, and relocation. Additionally, to solve large-sized instances, we devise a solution procedure based on the strategy of decomposing a difficult-to-solve problem into a set of easier-to-solve problems. The result of numerical experiments verifies the benefit of dynamic block stacking versus static block stacking by its lower operating cost and reduction in required floor space. It also shows, for practical-sized instances, the heuristic solution procedure developed surpasses solutions obtained using CPLEX in terms of computation time and solution quality.

We contribute (i) the first model for block stacking multiple products with changeable row depths, (ii) a solution method based on a strategy of decomposing the original problem into smaller and easier-to-solve sub-problems and, (iii) the quantitative analysis of the block stacking storage system's behavior when changing row depths by relocation is allowed.

The remainder of the paper is organized as follows. In Section 2, block stacking literature is reviewed and previous papers are characterized by common features. Section 3 addresses the fundamental assumptions of BSMPwRuDD and mathematically formulates it as a variation of the network-flow-based integer program. Section 4 introduces a solution method consisting of an upper bound procedure based on a time horizon decomposition heuristic and a lower bound procedure based on a Lagrangean decomposition heuristic. Based on the results of numerical experiments, Section 5 validates the solution procedure developed, verifies the benefit of DBS over SBS, and provides a comprehensive insight into the relocation behavior of a block stacking storage system adopting changeable row depths. Section 6 draws conclusions from the research and provides recommendations for future studies.

2. Literature Review

This section reviews block stacking storage research literature. Although block stacking is also common within an intermodal container terminal, we focus solely on its application within a warehouse. Container terminals employ different storage rules, such as allowing consolidation of unit loads of diverse groups in a stack and in a row. For more information regarding the block stacking storage method used within a container terminal, we refer the reader to Carlo *et al.* (2014). Previous literature reviews of the block stacking storage method in a warehouse can be found in Ashayeri and Gelders (1985) and Gu *et al.* (2010).

Table 2.1 summarizes papers that consider the problem of optimizing row depths, and further characterizes those papers according to solution, inventory profile, assumptions, objective function, and solution procedure. We found 19 block stacking storage papers published between 1961 and 2017.

In Table 2.1, the “solution” column categorizes the output of each paper's optimization problem using the following abbreviations

- SL single row depth for layout satisfying a given storage population
- SS single row depth for a single product lot
- SM single row depth for multiple product lots
- MS multiple row depths for a single product lot
- MM multiple row depths for multiple product lots
- CS changeable row depth for a single product lot
- CM changeable row depths for multiple product lots

Lee *et al*. (2016) is the first and only published paper studying changeable row depths.

The “inventory profile” column in Table 1 indicates if the inventory profile is assumed to be at the aggregate-level or the granular-level. With an aggregate-level inventory profile, in computing the value of the objective function, the inventory level is given as a fixed value, such as the maximum inventory level or average inventory level; with a granular-level inventory profile, fluctuations of inventory level over the time horizon are considered by using the cumulative number of rows occupied over the storage life of the product lot. The feature of inventory profile is abbreviated as follows:

- A aggregate-level inventory profile
- GD granular-level inventory profile assuming deterministic demand
- GS granular-level inventory profile assuming stochastic demand

Table 2.1: Characterization of articles by solution type, solution procedures, objective function considerations, and assumptions

Paper	Solution	Inventory Profile	Scope	Objective Function	Solution Procedure	Note
This paper	CM	GD	HN	FM	M, H	
Thorton (1961)	SL	A	NN	FN	E	
Hemmi (1963)	SL	A	NN	FN	E	
Kind (1965, 1975)	SS	GD	HN	FN	E, D	
Moder and Thornton(1965)	SL	A	NN	FN	E	
Berry (1968)	SL	A	HN	VM	D	Conventional aisle configuration
	SL	A	HN	VM	D	Diagonal aisle configuration
Roberts (1968)	MM	A	HN	FN, FM, VN	M	One row per one SKU, Variable block length
	MM	A	HN	FN, FM, VN	H	One row per one SKU, Fixed block length
	MM	A	HN	FN, FM, VN	M	Multi-rows per one SKU, Variable block length
	MM	A	HN	FN, FM, VN	M	Multi-rows per one SKU, Fixed block length
Kooy (1981)	MS	GD	HL	FN	E	
Matson (1982)	SS	GD	HN	FN	E, D	
	SS	GD	HN	FM	E	
	SM	GD	HN	FN	E	
	MS	GD	HL	FN	M, H	
	MM	GD	HN	FN	M, H	
Rickles and Elliott (1985)	SL	A	HN	FN	E	
Goetschalckx and Ratliff (1991)	MS	GD	HL	FN	M, H	
	MM	GD	HL	FN	H	Pattern perfectly balanced warehouse
Larson *et al*. (1997)	MM	A	HN	FM	H	
Koster (2010)	SM	A	HN	FN	D	
White et al. (2013)	SS	A	HN	FM	E	Dedicated storage
	SS	GD	HN	FM	E	Random storage
	MS	GD	HL	FM	E	
Bartholdi and Hackman (2014)	SM	GD	HN	FN	D	
Matson *et al*. (2014)	MS	A	HL	FM	M	Dedicated storage
	MS	GD	HL	FM	M	Random storage
Sonnentag *et al*. (2014)	SM	A	HN	FM	E, D	Dedicated storage
	SM	GD	HN	FM	E, D	Random storage
	MM	A	HL	FM	E, M	Dedicated storage
	MM	GD	HL	FM	E, M	Random storage
Kay (2015)	SM	A	HN	FN	D	Dedicated storage
	SM	A	HN	FN	D	Random storage
	SM	A	HN	VN	D	Dedicated storage
	SM	A	HN	VN	D	Random storage
Lee et al. (2016)	CS	GS	HN	FM	M	
Derhami *et al*. (2017)	SM	GD	HN	VN	D	Finite production rate

The “scope” column in Table 1 addresses if honeycomb loss and/or lot splitting is incorporated in the mathematical model. Even though honeycomb loss is a characteristic of block stacking storage, some early models omitted it in computing space utilization of a layout; space utilization was defined as the ratio of the designed storage area to total space, including a service area such as an aisle. In the literature, honeycomb loss was implicitly considered by the integrality property of the number of rows required or explicitly calculated based on unused time-space in a storage row. In some cases, it is simply approximated with an assumption. Splitting a lot into different row depths is another critical consideration in determining optimal row depths; if a product lot can be split among two or more row depths, it makes the problem more difficult to solve. In many papers, heuristics methods were proposed to obtain solutions. In many papers, heuristic methods were proposed to solve the problem of block stacking with lot splitting. In early papers, lot splitting occasionally indicated a storage policy of storing unit loads of a product lot, not in a single storage row, but in storage rows having the same depth. To represent which assumption is made, we use the following abbreviations.

- HL both honeycomb loss and lot splitting
- HN only honeycomb loss
- NL only lot splitting
- NN neither honeycomb loss nor lot splitting

The “objective function” column of Table 2.1 indicates the factors involved in each paper's objective function. This column shows either floor space or volume space is represented in the objective function, and if material handing cost is or is not considered in the objective function. When both space and material handling are considered, they are generally represented monetarily. Entries in this column include the following abbreviations.

- FM floor space and material handling
- FN only floor space
- VM volume space and material handling
- VN only volume space

The “solution approach” column in Table 2.1 indicates which solution procedure is used, recommended, and/or adopted to find a solution. The solution procedure is categorized as using enumeration (E), differentiation (D), mathematical programming (M), or heuristics (H). An enumeration approach finds an optimal row depth for a lot by considering all feasible solutions. Some papers relax the row depth’s integrality and identify a row depth that satisfies optimality conditions derived from differentiating the resulting cost function. A mathematical programming approach uses an optimization model and a technique such as dynamic programming or binary integer programming to obtain an optimal solution. Heuristics seek to obtain, with reasonable computational effort, a cost-efficient, but possibly suboptimal, row depth.

The “Note” column in Table 2.1 contains unique aspects of a paper. When multiple models are developed from one paper and cannot be categorized using the designations in the table, the entry in the column addresses any unique characteristics and clarifies the difference(s) among them.

Excluding Lee *et al*. (2016), all studies in Table 2.1 assume the row depth for a lot is permanent and did not consider changing the row depth by employing relocation. Interestingly, a few papers investigated the block stacking storage method from an operational viewpoint (Marsh, 1979; Derhami *et al*. 2016) using simulation. None of these papers allows the row depth for a lot to change during its storage life.

Our work is distinct from all previous studies by allowing changeable row depth for multiple product lots. Existing papers focus on finding an optimal row depth over a time horizon and,

consequently, an optimal design of the system; whereas, this paper seeks an optimal row depth each day over a time horizon and, as a result, seeks an optimal operating plan for the system. Compared to conventional research of block stacking, this study employs a more complicated mathematical model due to a larger number of decision variables.

3. Mathematical Model

In this section, we describe the mathematical model of BSMPwRuDD in determining an optimal DBS plan by specifying which product lot is stored in which storage area for each day over a time horizon. The model's objective is to minimize total cost incurred by daily operations of DBS: storage, replenishment, retrieval, and relocation. The cost of each is referred to as floor space cost, replenishment cost, retrieval cost, and relocation cost, respectively. Details regarding the costs and how they are calculated are provided in APPENDIX A of the dissertation. The notation of Table 2.2 is considered in this section.

Table 2.2: Notations for BSMPwRuDD

Notation	*Description*
L, l	number of lots considered, index of lot
T^l, t	cycle of inventory level profile of single product lot, index of day
T	cycle of inventory level profile of multiple product lots, $T = LCM(T^1, T^2, \dots, T^L)$
R	number of storage areas considered
q, r	index of present storage area, index of selected storage area
z^l	height of the stack of lot, measured in unit loads
Q^l	order quantity of lot l, measured in unit loads
D^l	daily demand of lot, measured in unit loads
I_t^l	inventory level of lot l at the beginning of day t, measured in unit loads,
d_r	depth of storage area r, measured in unit loads
P_r	the number of row positions in storage area r

Let $I_t^l, t = 1, \dots, \sigma^l T^l$, denote the inventory level of product l at the beginning of day t, where σ^l denotes the number of rotations of product l during the planning horizon. Without loss of generality, we assume $\sigma^l T^l = \sigma^{l'} T^{l'}$ for all pairs of products l and l', where the values σ^l can be determined by dividing the least common multiple of $\{T^l : l = 1, \dots, L\}$ by T^l. Let $T = \sigma^1 T^1 = \sigma^2 T^2 = \cdots = \sigma^L T^L$ denote the common planning horizon. By assuming the inventory levels for each lot l repeat on a T^l-day cycle, the collection $\{I_t^l : l = 1, \dots, L\}$ repeats on a T-day cycle.

By the definition of D^l and Q^l, there is no out of stock during the business hours and no back-order. We assume product lot l is reordered such that when the inventory level is zero at the end of business hours of a day, Q^l unit loads are replenished before starting business hours of the next day. Thus, given $\{I_1^l : l = 1, \dots, L\}$, each lot's subsequent inventory levels in periods t = 2,…, T can be determined as

$$I_t^l = \begin{cases} I_{t-1}^l - D^l, & \text{if } I_{t-1}^l - D^l > 0, \\ Q^l, & \text{otherwise.} \end{cases} \tag{2.1}$$

Because $Q^l = T^l D^l$, the inventory levels for product l will satisfy

$$I_t^l = I_{kT^l + t}^l, \quad \forall l = 1, \dots, L, \forall t = 1, \dots T^l, \forall k = 1, \dots, \sigma^l - 1, \tag{2.2}$$

and

$$I_1^l = \begin{cases} I_T^l - D^l, & \text{if } I_T^l - D^l > 0, \\ Q^l, & \text{otherwise.} \end{cases} \tag{2.3}$$

To find an optimal DBS plan given $\{I_t^l : l = 1, \dots, L; t = 1, \dots, T\}$, BSMPwRuDD is modeled as a network-flow-based integer program. In the directed network, a node indicates a storage area specified by a row depth and a day. The d_r-deep storage area at day t is represented as node $tR + r$ where $t = 0, \dots, T$ and $r = 1, \dots, R$. The node set, N, is defined as follows:

$$N = \{tR + r | t = 0, \dots, T, \quad r = 1, \dots, R\} \cup \{s, e\} \tag{2.4}$$

where nodes s and e indicate the start node and end node, respectively. For $t = 0, \dots, T$, let $N_t = \{tR + r | r = 1, \dots, R\}$ denote the set of nodes corresponding to day t. For convenience of exposition, we also define $N_{-1} = \{s\}$.

In developing the directed graph of BSMPwRuDD, t = 0 represents the last day of the previous T-day cycle of $\{I_t^l : l = 1, \dots, L\}$ and thus, $I_0^l = I_T^l$, l = 1,…,L. Because relocation cost is computed by the assigned storage area at the previous day and the present day, the nodes in N_0 are required to exactly express relocation cost at day 1 in the mathematical model of BSMPwRuDD. Additionally, by equating the assigned storage area at day 0 and day T, the mathematical model guarantees a cyclic solution repeats on a T-day cycle.

A directed arc indicates a decision of selecting a storage area for day t given the storage area chosen for day t-1. The arc originating from node $(t\text{-}1)R+q$ to node $tR+r$ denotes the decision of storing a lot in a d_r-deep storage area for day t after it is stored in a d_q-deep storage area on day t-1. From a node $tR+r$, R arcs emanate to all nodes in N_{t+1} where $t = 0, \dots, T$-1 and $r = 1, \dots, R$. The set of arcs, A, is defined as follows:

$$A = \{(tR + q, (t + 1)R + r) | t = 0, \dots, T - 1,\ \ q = 1, \dots, R,\ \ r = 1, \dots, R\} \cup \{(s, r) | r = 1, \dots, R\} \cup \{(TR + r, e) | r = 1, \dots, R\} \cup \{(e, s)\}. \quad (2.5)$$

The number of elements in A is calculated by $O(TR^2)$.

For each node $i \in N$, let $\mathrm{RN}(i) = \{j \in N | (j, i) \in A\}$ denote the set of nodes that have an outgoing arc to node i, and let $\mathrm{FN}(i) = \{j \in N | (i, j) \in A\}$ denote the set of nodes to which node i has an outgoing arc.

Figure 2.2 shows a directed network of BSMPwRuDD in which $R = T = 2$. Node 1 represents the d_1-deep storage area on day 0 and Node 4 indicates the d_2-deep storage area on day 1. Arc (3, 6) expresses the decision of selecting the d_2-deep storage area on day 2 when the d_1-deep storage area is chosen on day 1. At node 4, RN(4) is {1,2} and FN(4) is {5,6}.

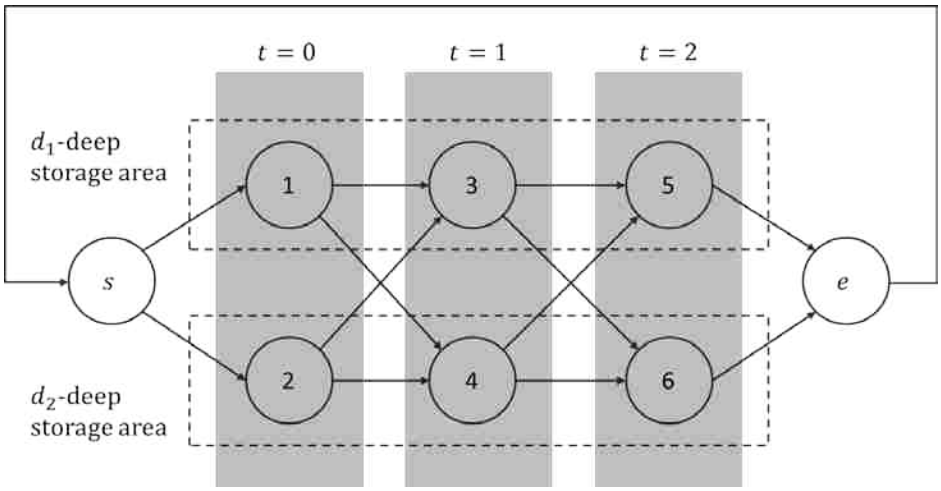

Figure 2.2: Instance of directed network of BSMPwRuDD where R=2 and T=2

Table 2.3: Notation for the BSMPwRuDD optimization problem

Notation	*Description*
Parameters:	
$c_{i,j}^{l}$	cost of arc (i,j) for lot l
n_j^l	usage of node capacity when lot l goes through node j
$\mathcal{C}_j$	capacity of node j
Decision variable:	
$x_{i,j}^{l}$	1 if arc (i,j) is selected for lot l, 0 otherwise

The notation in Table 2.3 is used in formulating the optimization problem. The value of $c_{i,j}^{l}$ for arc (i,j)=$\big((t\text{-}1)*R+q,\ t*R+r\big)$ denotes lot l's daily operating cost for day t when lot l is stored in a d_q-deep storage area for day t-1 and a d_r-deep storage area for day t. APPENDIX A develops the daily operating cost model of a single product lot, given by

$$OC = FS + ST + RT + BT, \tag{2.6}$$

where

$OC =$ daily operating cost of a single product lot,

$FS =$ floor space cost of a single product lot,

$ST =$ replenishment cost of a single product lot,

$RT =$ retrieval cost of a single product lot, and

$BT =$ relocation cost of a single product lot.

Given daily demand D^l, present storage area q, and assigned storage area r, the costs are easily redefined as a function of inventory level I_t^l as follows:

$$OC(I_t^l) = FS(I_t^l, r) + ST(I_t^l, r) + RT(I_t^l, D^l, r) + BT(I_t^l, q, r) \tag{2.7}$$

Using the daily operating cost function of I_t^l, $c^l_{(t-1)*R+q,t*R+r}$ is defined as according to

$$c^l_{(t-1)*R+q,t*R+r} = OC(I_t^l), \;\; t = 1, \dots, T, \;\; q = 1, \dots, R, \;\; r = 1, \dots, R, \tag{2.8}$$

$$c^l_{s,r} = 0 \text{ and } c^l_{TR+r,e} = 0, \quad r = 1, \dots, R, \tag{2.9}$$

and

$$c^l_{e,s} = 0. \tag{2.10}$$

The value of n_j^l at node j=t*R+r corresponds to the required number of row positions if lot l is stored in the d_r-deep storage area for day t. It is computed by

$$n^l_{t*R+r} = \left\lceil \frac{I_t^l}{d_r z^l} \right\rceil. \tag{2.11}$$

The value of $\mathcal{C}_j$ at node j=t*R+r expresses the number of row positions in the d_r-deep storage area. Specifically,

$$\mathcal{C}_{t*R+r} = P_r \tag{2.12}$$

Let $x^l_{(t-1)*R+q,t*R+r}$ equal one if product lot l is stored in the d_q-deep storage area for day t-1 and in the d_r-deep storage area for day t; otherwise, let $x^l_{(t-1)*R+q,t*R+r}$ equal zero.

The BSMPwRuDD optimization problem, hereafter referred to as IP-BSMPwRuDD, is formulated as a multi-commodity flow problem as follows:

IP-BSMPwRuDD:

$$\min \sum_{l=1}^{L} \sum_{(i,j)\in A} c_{ij}^{l}\, x_{ij}^{l} \tag{2.13}$$

subject to

$$\sum_{l=1}^{L} \sum_{i\in \mathrm{RN}(j)} n_j^l x_{ij}^l \le C_j, \quad \forall j \in N \setminus \{s, e\} \tag{2.14}$$

$$\sum_{i\in \mathrm{RN}(j)} x_{ij}^l - \sum_{k\in \mathrm{FN}(j)} x_{jk}^l = 0, \;\; j \in N \text{ and } l = 1, \dots, L \tag{2.15}$$

$$x_{es}^l = 1, \;\; l = 1, \dots, L \tag{2.16}$$

$$x_{s,i}^l = x_{T*R+i,e}^l, \quad i = 1, \dots, R \text{ and } l = 1, \dots, L \tag{2.17}$$

$$x_{ij}^l \in \{0,1\}, \quad \forall (i,j) \in A \text{ and } l = 1, \dots, L \tag{2.18}$$

Objective function (2.13) minimizes the sum of the product of the decision variable and the arc cost over all product lots and all arcs. The value of the objective function represents the total cost incurred by all product lots over T days. Constraint (2.14) is a node capacity constraint; it forces the sum of the flows on arcs incident to node j to be less than or equal to node capacity C_j and guarantees a dynamic block stacking plan defined by the solution satisfies the storage capacity constraint. Constraint (2.15) is a general flow balance constraint making a single product lot's supply and demand identical at each node; it ensures, in the dynamic block stacking plan, only one storage area is chosen for a single product lot at each day. Constraint (2.16) generates a flow of one unit for all product lots. Constraint (2.17) forces the chosen storage area for product lot l to be identical on days t = 0 and $t = T$. Constraint (2.18) prohibits lot splitting by requiring the x-variables to take on binary values.

4. Solution Procedure

In Section 3, BSMPwRuDD is modeled as a variation of the multi-commodity flow problem assuming unsplittable flow. Unlike the fractional multi-commodity flow problem solvable in polynomial time, the integral multi-commodity flow problem is NP-hard (Peinhardt, 2003). Thus, an efficient solution procedure is required to solve a practical-sized instance of BSMPwRuDD.

In this section, we develop a solution procedure adopting the strategy of decomposing the original problem into smaller and easier-to-solve sub-problems. We refer to the solution procedure as Decomposition Heuristics (DH). Figure 2.3 illustrates DH consisting of two parts: an upper bound procedure and a lower bound procedure.

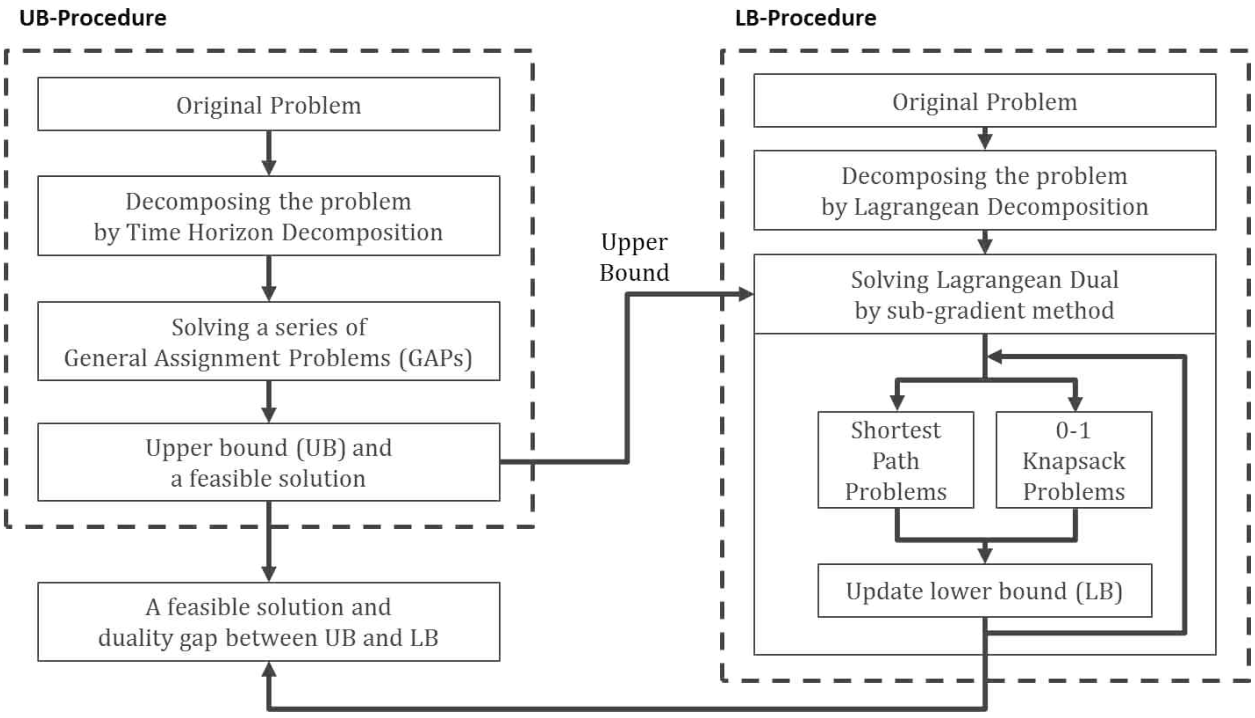

Figure 2.3: Summary of the solution procedure of Decomposition Heuristics

In the upper bound procedure, IP-BSMPwRuDD is decomposed into a series of single-day subproblems using a procedure we refer to as the *Time Horizon Decomposition* (THD) scheme.

With a given feasible solution of the previous day, the single-day subproblem reduces to a Generalized Assignment Problem (GAP). By sequentially combining solutions of single-day GAPs, we can establish a feasible solution and, consequently, compute the upper bound on the optimal objective function value of BSMPwRuDD.

In the lower bound procedure, IP-BSMPwRuDD decomposes, for a fixed assignment of values to dual variables, into a set of shortest path problems and a set of 0-1 knapsack problems by the Lagrangean Decomposition (LD) scheme. By applying a sub-gradient method to search the dual variable space, the LD heuristic iteratively improves the lower bound on an optimal objective function value of IP-BSMPwRuDD.

4.1. UB procedure: Time horizon decomposition heuristic

The THD heuristic disaggregates IP-BSMPwRuDD into subproblems in which product lots are assigned to storage areas for a single day. At first, THD solves the subproblems over an extended time horizon of ωT days (where $\omega \geq 1$ is an integer-valued parameter), gathering their solutions. Then, it builds feasible solutions of the original problem by combining the sub-problems' solutions. In other words, THD first determines daily DBS plans over an extended time horizon and then, develops a *united DBS plan* for the original time horizon by aggregating the daily DBS plans. By searching over an extended time horizon beyond the original time horizon, the THD heuristic establishes more candidates for the solution and increases the likelihood of finding a better solution.

In THD, three kinds of sub-problems are developed with different objectives. The first subproblem corresponding to day $\hat{t}$ is developed with the assumption the daily DBS plan of day $\hat{t}$-1 is defined in advance and determines a daily DBS plan of day $\hat{t}$. The second subproblem corresponding to day $\hat{t}$ is formulated with the supposition of the daily DBS plan of day $\hat{t}$-1 and

day $\hat{t}$+1 are decided beforehand and determines a daily DBS plan of day $\hat{t}$ guaranteeing a feasible united DBS plan. The third subproblem corresponding to day $\hat{t}$ is defined with the assumption of the predetermined daily DBS plan of day $\hat{t}$-1 and day $\hat{t}$+1. If possible, it updates a daily DBS plan of day $\hat{t}$ so that the operating cost of a united DBS plan decreases.

We first provide details of the first, second, and third subproblems in Section 4.1.1, Section 4.1.2, and Section 4.1.3. Then, in Section 4.1.4, we explain how THD works with the three sub-problems. In the following sections, we use the notation in Table 2.4. The attachment "(t)" specifies the set of elements of the notation corresponding to arcs from nodes in N_{t-1} to nodes in N_t. For example, $\mathbf{x}(t)$ represent the set of elements of $\mathbf{x}$ corresponding to arcs joined to nodes in N_t.

Table 2.4: Notations of variables of IP-BSMPwRuDD

Notation	*Description*
$\mathbf{x}$	subset of the variables of IP-BSMPwRuDD
$\bar{\mathbf{x}}$	fixed assignment to some $\mathbf{x}$
$\mathbf{X}$	all variables of IP-BSMPwRuDD
$\bar{\mathbf{X}}$	feasible solution of IP-BSMPwRuDD.

4.1.1. First sub-problem

The first sub-problem determines a daily DBS plan for day $\hat{t} = 0, \dots, \omega T$, assuming a daily DBS plan for day $\hat{t}$-1 is determined in advance. We define $\hat{t}^m$ as follows:

$$\hat{t}^m = \begin{cases} \hat{t} \bmod T, & \text{if } \hat{t} \geq 0 \text{ and } \hat{t} \bmod T > 0 \\ T, & \text{if } \hat{t} \geq 0 \text{ and } \hat{t} \bmod T = 0 \\ \hat{t}, & \text{if } \hat{t} < 0 \end{cases} \tag{2.19}$$

Let $\mathbf{x}(\hat{t}), \hat{t} = -1,0,1, \dots, \omega T$, indicate the subset of variables (of IP-BSMPwRuDD) corresponding to day $\hat{t}$, i.e.,

$$\mathbf{x}(-1) = \{x_{es}^l\}, \tag{2.20}$$

$$\mathbf{x}(0) = \left\{x_{sj}^{l} \middle| j \in N_0 \,;\, l = 1, \ldots, L\right\}, \text{ and} \tag{2.21}$$

$$\mathbf{x}(\hat{t}) = \left\{x_{ij}^{l} \middle| j \in N_{\hat{t}^m}; i \in \mathrm{RN}(j); l = 1, \ldots, L\right\} \text{ for } \hat{t} \geq 1. \tag{2.22}$$

For $l = 1, \ldots, L$, let $\mathbf{x}^l(\hat{t})$ represent the subset of variables of $\mathbf{x}(\hat{t})$ corresponding to product lot l, and let $x_{ij}^{l}(\hat{t})$ denote the variable of $\boldsymbol{x}^l(\hat{t})$ corresponding to arc (i,j) where $j \in N_{\hat{t}^m}$ and $i \in \mathrm{RN}(j)$.

Let $\bar{\mathbf{x}}(\hat{t})$, $\hat{t}$ = -1, 0, 1,…, ωT, indicate an assignment of values to $\mathbf{x}(\hat{t})$ and let $\bar{x}_{ij}^{l}(\hat{t})$ represent the element of $\bar{\mathbf{x}}(\hat{t})$ corresponding to arc (i,j) and product lot l. Due to Constraint (2.16), $x_{es}^{l}(-1)=1$ for any feasible solution to IP-BSMPwRuDD, and we therefore define $\bar{x}_{es}^{l}(-1)=1$.

We define the daily operating cost function at day $\hat{t} \in \{-1,0,1, \ldots, \omega T\}$, $f\left(\mathbf{x}(\hat{t})\right)$, as

$$f\left(\mathbf{x}(\hat{t})\right) = \begin{cases} \sum_{l=1}^{L} \sum_{j \in N_{\hat{t}^m}} \sum_{i \in \mathrm{RN}(j)} c_{ij}^{l} x_{ij}^{l}(\hat{t}), & \hat{t} \geq 1, \\ \sum_{l=1}^{L} \sum_{j \in N_0} \bar{c}_{sj}^{l} x_{sj}^{l}(0), & \hat{t} = 0, \end{cases} \tag{2.23}$$

where $\bar{c}_{sj}^{l}$ is defined by

$$\bar{c}_{sj}^{l} = FS\left(I_t^l, j\right) + ST\left(I_t^l, j\right) + RT\left(I_t^l, D^l, j\right). \tag{2.24}$$

The first sub-problem, referred to as THD$^1(\hat{t})$, optimizes over $\mathbf{x}(\hat{t})$ given fixed $\bar{\mathbf{x}}(\hat{t}\text{-}1)$ and is formulated for $\hat{t} = 0,1, \ldots, \omega T$ as follows:

THD$^1(\hat{t})$:

$$\min f\left(\mathbf{x}(\hat{t})\right) \tag{2.25}$$

subject to

$$\sum_{l=1}^{L} \sum_{i \in \mathrm{RS}(j)} n_j^l x_{ij}^{l}(\hat{t}) \leq C_j, \quad \forall j \in N_{\hat{t}^m} \tag{2.26}$$

$$\sum_{j \in N_{\hat{t}^m}} \sum_{i \in \mathrm{RS}(j)} x_{ij}^{l}(\hat{t}) = 1, \quad l = 1, \ldots, L \tag{2.27}$$

$$x_{ij}^{l}(\hat{t}) \leq \sum_{i \in N_{(\hat{t}-1)^m}} \sum_{h \in \mathrm{RS}(i)} \bar{x}_{hi}^{l}(\hat{t}-1), \quad \forall\, j \in N_{\hat{t}^m}, \forall i \in \mathrm{RS}(j), \text{and } l = 1, \ldots, L \tag{2.28}$$

$$x_{ij}^{l} \in \{0,1\}, \quad \forall j \in N_{\hat{t}^m}, \forall i \in \mathrm{RS}(j), \text{and } l = 1, \ldots, L \tag{2.29}$$

Constraint (2.26) ensures the daily DBS plan for day $\hat{t}$ satisfies each storage area's capacity. Constraint (2.27) limits each lot to be allocated to only one storage area in day $\hat{t}$. Constraint (2.28) restricts the domain of feasible solutions when $\bar{\mathbf{x}}(\hat{t}\text{-}1)$ is given. For example, if node i is selected for lot l in $\bar{\mathbf{x}}(\hat{t}\text{-}1)$, corresponding RHS of constraint (2.28) is one, otherwise zero. Thus, only arcs emanating from the node selected in $\bar{\mathbf{x}}(\hat{t}\text{-}1)$ are considered as feasible candidates for lot l among all arcs incident to nodes in N_t. Constraint (2.29) prevents lot splitting by having the x-variables take on binary values.

With these constraints, $\mathrm{THD}^1(\hat{t})$ is formulated as a GAP. Assume $\bar{\mathbf{x}}(\hat{t}\text{-}1)$ indicates product lot l is assigned to the storage area corresponding to node i^l at day $\hat{t}$-1. By constraint (2.28), $\mathbf{x}^l(\hat{t})$, l=1,..,L, are respectively reduced to $\{x_{ij}^l | j \in \mathrm{RN}(i^l)\}$ in which each variable corresponds to a node one-to-one. Constraint (2.26) represents the capacity of each node and constraint (2.27) limits each lot to be allocated to only one node. Thus, with the variable set reduced by constraint (2.28), solving $\mathrm{THD}^1(\hat{t})$ is identical to finding an assignment of L kinds of items (lots) to R bins (nodes or storage areas) minimizing cost, satisfying the capacity constraint.

For $1 \leq \hat{t} \leq T$, consider $\mathrm{THD}^1(\hat{t})$ and $\mathrm{THD}^1(\hat{t}+\alpha T)$, α = 1,…, ω-1. Notice $(\hat{t}+\alpha T)^m$ equals $\hat{t}^m$ because

$$\begin{aligned} \hat{t} \bmod \alpha T &= [(\hat{t} \bmod T) + (\alpha T \bmod T)] \bmod T \\ &= [(\hat{t} \bmod T)] \bmod T = \hat{t} \bmod T. \end{aligned} \tag{2.30}$$

Thus, $\hat{t}$ and $\hat{t}+\alpha T$ refer to the same day in the horizon of the original problem and $\text{THD}^1(\hat{t})$ and $\text{THD}^1(\hat{t}+\alpha T)$ are the same problem as long as $\bar{\mathbf{x}}(\hat{t}\text{-}1)$ and $\bar{\mathbf{x}}(\hat{t}+\alpha T\text{-}1)$ represent the same assignment of product lots to storage areas. It is, however, not guaranteed $\bar{\mathbf{x}}(\hat{t}\text{-}1)$ and $\bar{\mathbf{x}}(\hat{t}+\alpha T\text{-}1)$ indicate the same assignment in sequentially solving $\text{THD}^1(\hat{t})$, $\hat{t}$ = 0, 1, …, ωT, and consequently, $\bar{\mathbf{x}}(\hat{t})$ and $\bar{\mathbf{x}}(\hat{t}+\alpha T)$ may be different even though $\hat{t}$ and $\hat{t}+\alpha T$ indicate the same day.

Consider an example of BSMPwRuDD with ten product lots and three storage areas. Figure 2.4 depicts $\text{THD}^1(7)$ on a directed graph. The set of thick arcs represents the domain of feasible solutions of product lot 1 defined by constraint (2.28) of $\text{THD}^1(\hat{t})$ when product lot 1 is stored in the d_2-deep storage area at day 6. From the viewpoint of product lot 1, $\text{THD}^1(7)$ can be interpreted as assigning product lot 1 to node 22, 23, or 24. Considering the domain of feasible solutions is defined similarly for all product lots, $\text{THD}^1(7)$ can be interpreted as assigning ten product lots to node 22, 23, and 24. Figure 2.4 illustrates the characteristic of $\text{THD}^1(7)$ as GAP.

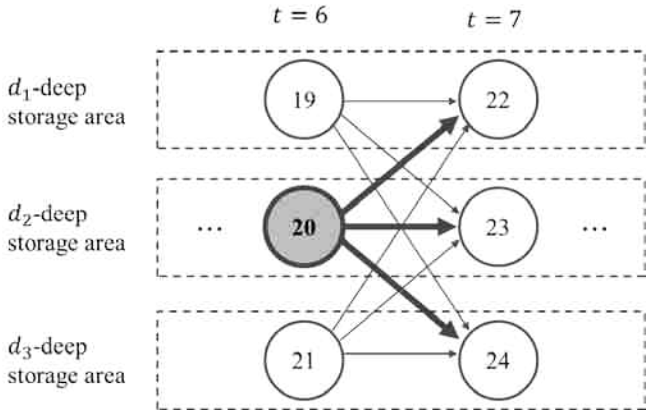

Figure 2.4: $\text{THD}(7)^-$ on a directed graph

4.1.2. Second sub-problem

For each $\hat{t} = T-1, T, \ldots, \omega T$, consider the partial solution of IP-BSMPwRuDD built by setting $\mathbf{x}(\bar{t}^m) = \bar{\mathbf{x}}(\bar{t})$ for $\bar{t}$ = $\hat{t}$-T+2, $\hat{t}$-T+3,…, $\hat{t}$-1. Fixing $\mathbf{x}(\bar{t}^m) = \bar{\mathbf{x}}(\bar{t})$ for $\bar{t}$ = $\hat{t}$-T+2, $\hat{t}$-T+3,…, $\hat{t}$-1

establishes each lot's storage area on both day $\bar{t}^m$-1 and day $\bar{t}^m$; thus, the partial solution assigns each lot to a storage area for each day, with the exception that none of the lots are assigned to a storage area on day $\hat{t}^m$.

To establish a (complete) feasible solution of IP-BSMPwRuDD, we define the second sub-problem, THD$^2(\hat{t})$, for $\hat{t}$ = T-1, T,…, ωTas the version of IP-BSMPwRuDD that results when $\mathbf{x}(\bar{t}^m) = \bar{\mathbf{x}}(\bar{t})$ is fixed for all $\bar{t}$ = $\hat{t}$-T+2, $\hat{t}$-T+3,…, $\hat{t}$-1. In this case, the variable set of IP-BSMPwRuDD is reduced to $\mathbf{x}(\hat{t}^m)$ and $\mathbf{x}((\hat{t}+1)^m)$. After solving THD$^2(\hat{t})$, let $\bar{\boldsymbol{x}}(\hat{t})$ and $\bar{\boldsymbol{x}}(\hat{t}+1)$ denote the optimal values of $\mathbf{x}(\hat{t})$ and $\mathbf{x}(\hat{t}+1)$.

Consider an example of BSMPwRuDD with two product lots and two storage areas. Figure 2.5 illustrates the set of decision variables for product lot 1 of THD$^2(\hat{t})$ for $\hat{t} = 11$ and $T = 4$. Assume $\bar{\mathbf{x}}^1(\hat{t}\text{-}T+2) = \bar{\mathbf{x}}^1(9)$ is given as $[\bar{x}^1_{1,3}(9), \bar{x}^1_{1,4}(9), \bar{x}^1_{2,3}(9), \bar{x}^1_{2,4}(9)] = [0, 1, 0, 0]$ and $\bar{\mathbf{x}}^1(\hat{t}\text{-}1) = \bar{\mathbf{x}}^1(10)$ is given as $\left[\bar{x}^1_{3,5}(10), \bar{x}^1_{3,6}(10), \bar{x}^1_{4,5}(10), \bar{x}^1_{4,6}(10)\right] = [0, 0, 1, 0]$.

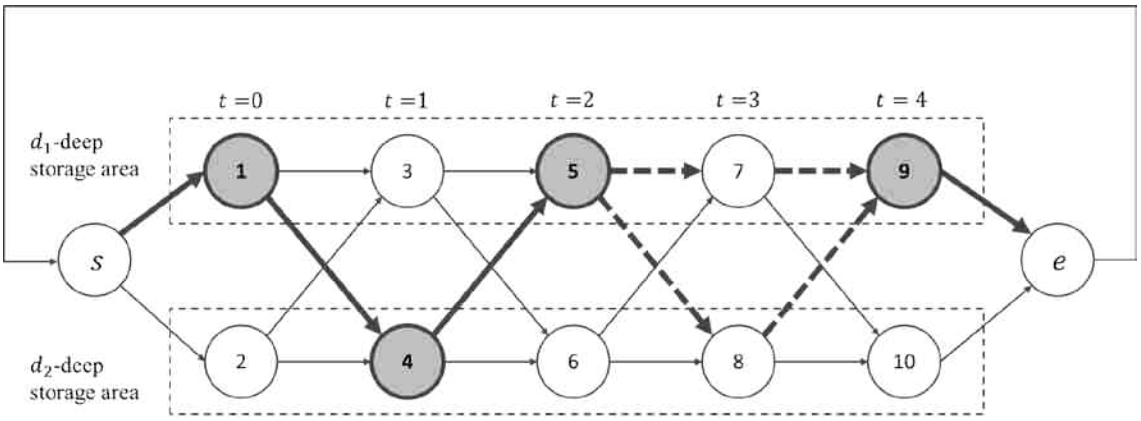

Figure 2.5: THD$^2(11)$ on a directed graph in which for product lot 1, the solid arcs represent fixed variables, dashed arcs indicate the feasible domain of variables, and the shaded nodes express the assigned storage area on day *t*.

Fixing $x^1_{1,4}(1) = \bar{x}^1_{1,4}(9)$ = 1 assigns product lot 1 to storage area 1 on day 0 and (by constraint (2.14)) day 4 and to storage area 2 on day 1. Fixing $x^1_{4,5}(2) = \bar{x}^1_{4,5}(10) = 1$ assigns product lot

1 to storage area 1 on day 2. Consequently, the variable set of product lot 1 of IP-BSMPwRuDD is reduced to $\mathbf{x}^1(3)$ and $\mathbf{x}^1(4)$. The set of dotted arcs emanating from node 5 represents the feasible domain of $\mathbf{x}^1(3)$ and the set of dashed arcs incident to node 9 indicates the feasible domain of $\mathbf{x}^1(4)$.

For $\hat{t} = T-1, T, \ldots, \omega T$, let $\overline{\mathbf{X}}_{\hat{t}}$ represent the solution of IP-BSMPwRuDD built by combining $\bar{\mathbf{x}}(\hat{t}\text{-}T\text{+}2)$, $\bar{\mathbf{x}}(\hat{t}\text{-}T\text{+}2),\ldots,\bar{\mathbf{x}}(\hat{t}\text{-}1)$, $\check{\mathbf{x}}(\hat{t})$, and $\check{\mathbf{x}}(\hat{t}\text{+}1)$. Let $\overline{\mathbf{X}}_{\hat{t}}(t)$, representing the elements of $\overline{\mathbf{X}}_{\hat{t}}$ corresponding to day t, be defined as

$$\overline{\mathbf{X}}_{\hat{t}}(\bar{t}^m) = \begin{cases} \bar{\mathbf{x}}(\bar{t}), & \text{for } \bar{t} = \hat{t}-T+2, \hat{t}-T+3, \ldots, \hat{t}-1, \\ \check{\mathbf{x}}(\bar{t}), & \text{for } \bar{t} = \hat{t}, \hat{t}+1. \end{cases} \tag{2.31}$$

Let $\left[\bar{X}_{ij}^l\right]_{\hat{t}}(t)$ represent the element of $\overline{\mathbf{X}}_{\hat{t}}(t)$ corresponding to arc (i,j) and product lot l. Given $\overline{\mathbf{X}}_{\hat{t}}$, x_{si}^l and x_{je}^l are defined as follows:

$$x_{si}^l = \sum_i \sum_{j \in \mathrm{FS}(i)} \left[\bar{X}_{ij}^l\right]_{\hat{t}}(t) \quad \text{for } i \in N_0, \tag{2.32}$$

and

$$x_{je}^l = \sum_j \sum_{i \in \mathrm{RS}(j)} \left[\bar{X}_{ij}^l\right]_{\hat{t}}(t) \quad \text{for } j \in N_T. \tag{2.33}$$

Let $F[\overline{\mathbf{X}}_{\hat{t}}]$ be the objective function value of IP-BSMPwRuDD, given a solution represented by $\overline{\mathbf{X}}_{\hat{t}}$. It is computed by

$$F[\overline{\mathbf{X}}_{\hat{t}}] = \sum_{t=1}^{T} f\left(\overline{\mathbf{X}}_{\hat{t}}(t)\right) \tag{2.34}$$

4.1.3. Third sub-problem

In Section 4.1.1 and Section 4.1.2, we show how to build $(\omega-1)T$+2 feasible solutions $\overline{\mathbf{X}}_{\hat{t}}$, $\hat{t} = T-1, T-2, \ldots, \omega T$, to IP-BSMPwRuDD. Even though $\overline{\mathbf{X}}_{\hat{t}}$ is a combination of optimal solutions

of THD$^1(\bar{t})$, $\bar{t}$ = $\hat{t}$-T+2, $\hat{t}$-T+3,…, $\hat{t}$-1 and THD$^2(\hat{t})$, $\overline{\mathbf{X}}_{\hat{t}}$ might be sub-optimal for the original problem.

Given a feasible solution $\mathbf{X} = \overline{\mathbf{X}}$ to IP-BSMPwRuDD, the third subproblem (denoted as "THD$^3(\overline{\mathbf{X}}, \hat{t})$") aims to identify an improved solution by modifying the lot-to-storage-area assignments on a single day, $\hat{t} = 1, \dots, T$. For a feasible solution $\overline{\mathbf{X}}$ to IP-BSMPwRuDD and $\hat{t} = 1, \dots, T$, let THD$^3(\overline{\mathbf{X}}, \hat{t})$ refer to the version of IP-BSMPwRuDD that results when $\mathbf{X}(\bar{t}) = \overline{\mathbf{X}}(\bar{t})$ is fixed for all $\bar{t} \in \{1, \dots, T\} \setminus \{\hat{t}, (\hat{t}+1)^m\}$. Noting that $\mathbf{X}(\hat{t})$ and $\mathbf{X}((\hat{t}+1)^m)$ are the only variables of IP-BSMPwRuDD that have not yet been fixed, solving THD$^3(\overline{\mathbf{X}}, \hat{t})$ yields a solution $\widehat{\mathbf{X}}(\hat{t})$ and $\widehat{\mathbf{X}}((\hat{t}+1)^m)$. If

$$f\left(\widehat{\mathbf{X}}(\hat{t})\right) + f\left(\widehat{\mathbf{X}}((\hat{t}+1)^m)\right) < f\left(\overline{\mathbf{X}}(\hat{t})\right) + f\left(\overline{\mathbf{X}}((\hat{t}+1)^m)\right), \tag{2.35}$$

an improved feasible solution $\widetilde{\mathbf{X}}$ to IP-BSMPwRuDD is obtained as

$$\widetilde{\mathbf{X}}(\bar{t}) = \begin{cases} \overline{\mathbf{X}}(\bar{t}), & \bar{t} \in \{1, \dots, T\} \setminus \{\hat{t}, (\hat{t}+1)^m\}, \\ \widehat{\mathbf{X}}(\bar{t}), & \bar{t} \in \{\hat{t}, (\hat{t}+1)^m\}. \end{cases} \tag{2.36}$$

4.1.4. Procedure of THD heuristic

The THD heuristic consists of two stages. In the first stage, THD$^1(\hat{t})$, $\hat{t}$ = 0,1,…,ωT, are solved sequentially, determining daily DBS plan $\bar{\mathbf{x}}(\hat{t})$. In the second stage, THD$^2(\hat{t})$, $\hat{t}$ = T-1, T,…, ωT, are solved, building united DBS plan $\overline{\mathbf{X}}_{\hat{t}}$. Once $\overline{\mathbf{X}}_{\hat{t}}$ is ready, by solving THD$^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$, $\bar{t}$ = 1, 2, …, T, $\overline{\mathbf{X}}_{\hat{t}}$ is updated such that $F[\overline{\mathbf{X}}_{\hat{t}}]$ is decreased. Define F^* and $\overline{\mathbf{X}}^*$ as follows:

$$F^* = \min_{\overline{\mathbf{X}}_{\hat{t}}, \hat{t} = T\text{-}1, T, \dots, \omega T} F[\overline{\mathbf{X}}_{\hat{t}}] \tag{2.37}$$

and

$$\overline{\mathbf{X}}^* \in \operatorname*{argmin}_{\overline{\mathbf{X}}_{\hat{t}}, \hat{t} = T\text{-}1, T, \dots, \omega T} F[\overline{\mathbf{X}}_{\hat{t}}] \tag{2.38}$$

Figure 2.6 describes the pseudo-code of the THD algorithm. Line (1), (2), and (3) are the first stage where THD$^1(\hat{t})$, $\hat{t}$ = 0,1,…,ωT, are solved sequentially, determining daily DBS plan $\bar{\mathbf{x}}(\hat{t})$. Lines from (4) to (31) are the second stage where THD$^2(\hat{t})$, $\hat{t}$ = T-1, T,…, ωT, are solved, building united DBS plan $\overline{\mathbf{X}}_{\hat{t}}$. Once $\overline{\mathbf{X}}_{\hat{t}}$ is ready by line (6) and (7), through from line (8) to (29), $\overline{\mathbf{X}}_{\hat{t}}$ is updated such that $F[\overline{\mathbf{X}}_{\hat{t}}]$ is decreased by solving THD$^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$, $\bar{t}$ = 1, 2, …, T. In line (30), the notation $\overline{\mathbf{X}}^{\mathrm{Cur}}$ and F^{Cur} indicates an incumbent solution and its objective function value. Consequently, line (30) finds $F^*$and $\overline{\mathbf{X}}^*$. At the termination of the algorithm, the THD returns $\overline{\mathbf{X}}^{\mathrm{Cur}}$ and F^{Cur} corresponding to $\overline{\mathbf{X}}^*$ and F^*, respectively.

Notice lines from (8) to (29) representing the procedure of updating $\overline{\mathbf{X}}_{\hat{t}}$. Given $\overline{\mathbf{X}}_{\hat{t}}$ at $\hat{t}$ = T-1, T, …, ωT, the algorithm sequentially solves THD$^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$ according to a systematic order of $\bar{t}$. The algorithm dynamically changes $\bar{t}$ back and forth according to a predetermined rule of guaranteeing all of each THD$^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$ for given $\overline{\mathbf{X}}_{\hat{t}}$ is solved to check if their solution improves $\overline{\mathbf{X}}_{\hat{t}}$. The algorithm sequentially solves THD$^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$ increasing $\bar{t}$ by one as long as the solution of THD$^3(\overline{\mathbf{X}}_{\hat{t}}, (\bar{t}-1)^m)$ improves $\overline{\mathbf{X}}_{\hat{t}}$. It is referred to as the forward-search. In addition, the algorithm consecutively solves THD$^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$ decreasing $\bar{t}$ by one as long as the solution of THD$^3(\overline{\mathbf{X}}_{\hat{t}}, (\bar{t}+1)^m)$ cannot improve $\overline{\mathbf{X}}_{\hat{t}}$. It is referred to as the backward-search. We use the notation $\phi(\bar{t})$ to indicate the problem status of THD$^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$. For $\bar{t}$ = 1,2,…,T, $\phi(\bar{t})$ is 1 if THD$^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$ needs to be revisited; 0 if THD$^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$ is not visited or not concluded; and -1 if THD$^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$ concludes its solution cannot improve $\overline{\mathbf{X}}_{\hat{t}}$. Initially, $\phi(\bar{t})$, $\bar{t}$ = 1,2,…,T are set as 0 in line (8). Based on predetermined conditions, $\phi(\bar{t})$ is set as 1 in line (14) and (16) and -1 in line (21) and (23). When each $\phi(\bar{t})$ is set as -1, the loop of updating $\overline{\mathbf{X}}_{\hat{t}}$ is terminated. Appendix A provides examples of the procedure of updating $\overline{\mathbf{X}}_{\hat{t}}$ to clarify the rule of changing $\bar{t}$ and the condition of setting the value of $\phi(\bar{t})$.

THD Algorithm:

FOR $\hat{t}$ = 0 to ωT (1)

Solve THD$^1(\hat{t})$ with $\bar{\mathbf{x}}(\hat{t}\text{-}1)$ and determine $\bar{\mathbf{x}}(\hat{t})$ (2)

END FOR (3)

Set F^{Cur} as a very big number. (4)

FOR $\hat{t}$ = T-1 to ωT (5)

Solve THD$^2(\hat{t})$ with $\bar{\mathbf{x}}(t)$, t = $\hat{t}$-T+2, $\hat{t}$-T+3,…, $\hat{t}$-1 and determine $\breve{\mathbf{x}}(\hat{t})$ and $\breve{\mathbf{x}}(\hat{t}+1)$ (6)

Define $\overline{\mathbf{X}}_{\hat{t}}$ with $\bar{\mathbf{x}}(t)$, t = $\hat{t}$-T+2, $\hat{t}$-T+3,…, $\hat{t}$-1 and $\breve{\mathbf{x}}(\hat{t})$ and $\breve{\mathbf{x}}(\hat{t}+1)$ (7)

Do $\bar{t} \leftarrow 1$, $SD \leftarrow$ Forward, and $\phi(t) \leftarrow 0$ for t = 1,2,…,T (8)

WHILE $\phi(t)$ is not -1 for t = 1,2,…,T (9)

Solve THD$^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$ and determine $\widehat{\mathbf{X}}(\bar{t})$ and $\widehat{\mathbf{X}}((\bar{t}+1)^m)$ (10)

IF $f\left(\widehat{\mathbf{X}}(\bar{t})\right) + f\left(\widehat{\mathbf{X}}((\bar{t}+1)^m)\right) < f\left(\overline{\mathbf{X}}_{\hat{t}}(\bar{t})\right) + f\left(\overline{\mathbf{X}}_{\hat{t}}((\bar{t}+1)^m)\right)$ (11)

Do $\overline{\mathbf{X}}_{\hat{t}}(\bar{t}) \leftarrow \widehat{\mathbf{X}}(\bar{t})$ and $\overline{\mathbf{X}}_{\hat{t}}((\bar{t}+1)^m) \leftarrow \widehat{\mathbf{X}}((\bar{t}+1)^m)$ (12)

IF *SD* is Forward (13)

Do $\phi((\bar{t}-1)^m) \leftarrow 1$, $\phi((\bar{t}+1)^m) \leftarrow 1$, and $\bar{t} \leftarrow (t+1)^m$ (14)

ELSE IF *SD* is Backward (15)

Do $\phi((\bar{t}-1)^m) \leftarrow 1$, $\phi((\bar{t}+1)^m) \leftarrow 1$, $SD \leftarrow$ Forward, and $\bar{t} \leftarrow (t+1)^m$ (16)

END IF (17)

ELSE (18)

IF *SD* is Forward (19)

IF $\phi(\bar{t})$ is not zero (20)

Do $\phi((\bar{t}-1)^m) \leftarrow -1$, $\phi(\bar{t}) \leftarrow -1$, $SD \leftarrow$ Backward, and $\bar{t} \leftarrow (\bar{t}-2)^m$ (21)

ELSE IF $\phi(\bar{t})$ is zero (22)

Do $\phi(\bar{t}) \leftarrow -1$ and $\bar{t} \leftarrow (\bar{t}+1)^m$ (23)

END IF (24)

ELSE IF *SD* is Backward (25)

Do $\phi(\bar{t}) \leftarrow -1$ and $\bar{t} \leftarrow (\bar{t}-1)^m$ (26)

END IF (27)

END IF (28)

END WHILE (29)

IF $F[\overline{\mathbf{X}}_{\hat{t}}] < F^{\text{Cur}}$, **Do** $F^{\text{Cur}} \leftarrow F[\overline{\mathbf{X}}_{\hat{t}}]$ and $\overline{\mathbf{X}}^{\text{Cur}} \leftarrow \overline{\mathbf{X}}_{\hat{t}}$ (30)

END FOR (31)

Return $\overline{\mathbf{X}}^{\text{Cur}}$ and F^{Cur} as the solution of BSMPwRuDD (32)

Figure 2.6: Pseudo-code of THD algorithm

4.2. LB procedure: LD heuristics

The lower-bounding procedure is motivated by the structure of IP-BSMPwRuDD, in which removal of the constraints (2.14) yields separable shortest path models. Based upon this structure, we develop a Lagrangean-decomposition-based lower-bounding procedure in which we decouple constraints (2.14) from the remaining constraints.

Lagrangean relaxation is a technique taking a set of complicating constraints into the objective function with Lagrangean multipliers (Geoffrion, 1974). The resulting Lagrangean problem is easier-to-solve compared to the original problem and its objective function value works as a lower bound on the original problem in case of the minimization problem (an upper bound in case of the maximization problem) (Fisher, 2004). Theoretically, the bound given by Lagrangean relaxation is at least as tight as the one given by linear programming relaxation. Applications of Lagrangean relaxation include solving the traveling salesman problem, the scheduling problem, the general IP problem, the location problem, the generalized assignment problem, the set covering-partitioning problem, and so on (Fisher, 2004).

The Lagrangean decomposition method is a special case of Lagrangean relaxation. Lagrangean decomposition is a technique relaxing linking constraints by introducing identical copies of the original decision variables. The constraints equating the copies and the original decision variables are added as new complicating constraints, making it possible to decompose the original problem into two or more sub-problems. The equating constraints are taken into the objective function with Lagrangean multipliers (Guignard and Rosenwein, 1990). Lagrangean decomposition keeps all the original constraints in the decomposed sub-problems; it yields bounds substantially better than or at least as tight as standard Lagrangean relaxation bounds (Guignard and Kim, 1987).

Because of the problem structure consisting of ease-to-solve problems complicated by a relatively small set of side constraints, it has been highly motivated for solving the multi-commodity flow problem to relax the complicating constraints and then decompose the original problem. When the complicating constraints are relaxed, the multi-commodity flow problem is reduced to smaller and easier-to-solve sub-problems such as linear or convex minimum cost flow problems or shortest path problems (Ouorou *et al*, 2000). The unsplittable multi-commodity flow problem can be decomposed into shortest path problems by relaxing the capacity constraint using Lagrangean relaxation. (Frangioni, 2005). See Ahuja *et al*. (1993) for an exemplary application of Largrangean relaxation in solving the multi-commodity flow problem.

Considering the proven result of applying Lagrangean relaxation in solving the multi-commodity flow problem in the literature and a possibility of better performance compared to Lagrangean relaxation, we adopt the Lagrangean decomposition method to solve IP-BSMPwRuDD and, at least, to produce a good lower bound.

To relax IP-BSMPwRuDD using the Lagrangean decomposition technique, we define new variables y_{ij}^l, l=1,..,L and $(i,j) \in A$, let y_{ij}^l replace x_{ij}^l in constraints (2.14) of IP-BSMPwRuDD, and insert constraints equating x_{ij}^l and y_{ij}^l. The resulting model is referred to as PreMP and defined as in the next page.

Finally, constraint (2.44) of PreMP is relaxed by introducing the unsigned Lagrangean multiplier, u_{ij}^l and adding the terms of $u_{ij}^l\left(x_{ij}^l - y_{ij}^l\right)$ to the objective function. The resulting objective function is referred to as the Lagrangean function, $L(\mathbf{x}, \mathbf{y}, \mathbf{u})$. Notice, $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{u}$ represent the vectors of x_{ij}^l, y_{ij}^l, and u_{ij}^l, respectively. The resulting relaxed problem of PreMP is referred to as MP and defined as in the next page.

PreMP:

$$\min \sum_{l=1}^{L} \sum_{(i,j)\in A} c_{ij}^{l}\, x_{ij}^{l} \tag{2.39}$$

subject to

$$\sum_{l=1}^{L} \sum_{(i,j)\in \bar{A}_j} n_j^l y_{ij}^l \le N_j, \quad \text{for } \forall j \in N \setminus \{s,e\} \tag{2.40}$$

$$\sum_{(i,j)\in \bar{A}_j} x_{ij}^l - \sum_{(j,k)\in \underline{A}_j} x_{jk}^l = 0, \;\; j \in N \text{ and } l = 1,\dots,L \tag{2.41}$$

$$x_{es}^l = 1, \;\; l = 1,\dots,L \tag{2.42}$$

$$x_{s,i}^l = x_{i+T*R,e}^l, \quad i = 1,\dots,R \text{ and } l = 1,\dots,L \tag{2.43}$$

$$x_{ij}^l = y_{ij}^l, \quad \text{for } \forall (i,j) \in A \text{ and } l = 1,\dots,L \tag{2.44}$$

$$x_{ij}^l \in \{0,1\}, \;\; \text{for } \forall (i,j) \in A \text{ and } l = 1,\dots,L \tag{2.45}$$

$$y_{ij}^l \in \{0,1\}, \;\; \text{for } \forall (i,j) \in A \text{ and } l = 1,\dots L \tag{2.46}$$

MP:

$$\min_{\mathbf{x},\mathbf{y}} L(\mathbf{x},\mathbf{y},\mathbf{u}) \;\; = \sum_{l=1}^{L} \sum_{(i,j)\in A} c_{ij}^{l}\, x_{ij}^{l} + \sum_{i=1}^{L} \sum_{(i,j)\in A} u_{ij}^{l}\left(x_{ij}^{l} - y_{ij}^{l}\right) \tag{2.47}$$

subject to

(2.40), (2.41), (2.43), (2.45), and (2.46) of Pre-MP

and

$$u_{ij} \in \mathbb{R}, \;\; \text{for } \forall (i,j) \in A \tag{2.48}$$

Set $Z_{LD1}(x, u)$ and $Z_{LD2}(y, u)$ as follows:

$$Z_{LD1}(\mathbf{x}, \mathbf{u}) = \sum_{l=1}^{L} \sum_{(i,j)\in A} \left(c_{ij}^{l} + u_{ij}^{l}\right) x_{ij}^{l} \tag{2.49}$$

$$Z_{LD2}(\mathbf{y}, \mathbf{u}) = \sum_{i=1}^{L} \sum_{(i,j)\in A} u_{ij}^{l} y_{ij}^{l}. \tag{2.50}$$

Then, the objective function of MP can be stated using $Z_{LD1}(\mathbf{x}, \mathbf{u})$ and $Z_{LD2}(\mathbf{y}, \mathbf{u})$ as follows:

$$\min_{\mathbf{x},\mathbf{y}} L(\mathbf{x}, \mathbf{y}, \mathbf{u}) = \min_{\mathbf{x},\mathbf{y}} \left(Z_{LD1}(\mathbf{x}, \mathbf{u}) - Z_{LD2}(\mathbf{y}, \mathbf{u})\right) = \min_{\mathbf{x}} Z_{LD1}(\mathbf{x}, \mathbf{u}) - \max_{\mathbf{y}} Z_{LD2}(\mathbf{y}, \mathbf{u}). \tag{2.51}$$

Consequently, the MP can be decomposed into two sub-problems, referred to as LD-Sub1 and LD-Sub2, as follows:

LD-Sub1**:**

$$\min_{\mathbf{x}} Z_{LD1}(\mathbf{x}, \mathbf{u}) = \sum_{l=1}^{L} \sum_{(i,j)\in A} \left(c_{ij}^{l} + u_{ij}^{l}\right) x_{ij}^{l} \tag{2.52}$$

subject to

(2.41), (2.43), and (2.45)of Pre-MP and (2.48) of MP

LD-Sub2:

$$\max_{\mathbf{y}} Z_{LD2}(\mathbf{y}, \mathbf{u}) = \sum_{i=1}^{L} \sum_{(i,j)\in A} u_{ij}^{l} y_{ij}^{l} \tag{2.53}$$

subject to

(2.40) and (2.46) of Pre-MP and (2.48) of MP

LD-Sub1 separates into $L*R$ shortest path problems. For each product lot, we need to solve one shortest path problem for each possible value of the first storage area. We can easily obtain an optimal solution of LD-Sub1 by simply aggregating an optimal shortest path of each product lot. LD-Sub2 is a set of knapsack problems corresponding to each day over the time horizon. It is

known as a weakly NP-hard problem and can be solved efficiently using dynamic programming. See Martello and Toth (1990) for more details of a dynamic programming-based algorithm to solve the knapsack problem.

Let X and Y indicate feasible regions of x_{ij}^{l} and y_{ij}^{l}, respectively. Then, the dual function of MP, $Z_{DLD}(\mathbf{u})$, is defined as follows:

$$Z_{DLD}(\mathbf{u}) = \min_{x \in X, y \in Y} L(\mathbf{x}, \mathbf{y}, \mathbf{u}) = \min_{x \in X} Z_{LD1}(\mathbf{x}, \mathbf{u}) - \max_{y \in Y} Z_{LD2}(\mathbf{y}, \mathbf{u}). \tag{2.54}$$

The resulting dual problem of MP is referred to as LD-Dual and defined as follows:

LD-Dual:

$$\max_{\mathbf{u}} \quad Z_{DLD}(\mathbf{u}) \tag{2.55}$$

$$\mathbf{u} \text{ is unsigned} \tag{2.56}$$

By the weak duality theorem, the value of $Z_{DLD}(\mathbf{u})$ provides a lower bound on the objective function value of an optimal solution of MP. Solving the Lagrangean dual problem, even approximately, produces a lower bound on the optimum objective function value of the original problem (Bertsekas, 2003). Because fortunately, LD-Sub1 and LD-Sub2 are not too hard to solve, the sub-problems of MP are solved exactly in the procedure of solving the LD-Dual to obtain a lower bound.

To solve LD-Dual, we use the hybrid sub-gradient algorithm of Guta (2003) with minor modifications. The initial upper bound is set as the upper bound established by THD heuristics and the initial lower bound is set as the optimal objective function value of linearly relaxed PreMP (LP-PreMP). The initial Lagrangean multiplier is set as the value of the dual variables corresponding to constraint (2.44) of PreMP when the problem is solved optimally with linearly relaxed primal variables. Consequently, the procedure of solving LP-PreMP is embedded in the

lower bound procedure as a preliminary step before starting the sub-gradient method. The idea of this algorithm is to repeatedly solve LD with fixed Lagrangean multipliers, $\hat{\mathbf{u}}$, and then to iteratively search for the multipliers $\hat{\mathbf{u}}$ yielding the tightest bound. The notation of Table 2.5 is used in introducing the pseudo-code of the hybrid sub-gradient algorithm. Tuning parameters, N^{MaxUI}, DecRate, and MaxIter, are set based on the preliminary experiment and the value of deflection angle regulator τ is set as 1.5, following Guta (2003).

Table 2.5: Notations of the Hybrid sub-gradient algorithm

Notation	*Description*
BKLB	acronym of Best Known Lower Bound
BKUB	acronym of Best Known Upper Bound
OFV	acronym of Objective Function Value
MaxIter	maximum number of iterations
DecRate	step length decreasing rate
N^{UI}	number of un-improvements at given step length
N^{MaxUI}	value of N^{UI} forcing step length reduction
γ	step length regulator
τ	deflection angle regulator
$\hat{\mathbf{u}}^k$	Lagrangean multiplier fixed at stage k
$\mathbf{x}^k$	optimal solution of LD-Sub1 given $\hat{\mathbf{u}}^k$
$\mathbf{y}^k$	optimal solution of LD-Sub2 given $\hat{\mathbf{u}}^k$
$\mathbf{s}^k$	sub-gradient at stage k
λ^k	step length at stage k
δ_k	deflection indicator at stage k
$\mathbf{\Delta}^k$	hybrid step direction at stage k

Figure 2.7 is the pseudo-code of the hybrid sub-gradient algorithm. For more details of the algorithm, refer to Guta (2003). See Fisher (1985) and Mao *et al*. (2015) for more details on the application of the sub-gradient method in solving the Lagrangean dual problem.

Hybrid sub-gradient Algorithm:

Solve LP-PreMP and set initial Lagrangean multiplier $\hat{\mathbf{u}}^1$ (1)

Do BKLB ← OFV of LP-PreMP and BKUB ← F^* (2)

Do $N^{\mathrm{UI}} \leftarrow 0$, $N^{\mathrm{MaxUI}} \leftarrow 0$, DecRate ← 0.9, and MaxIter ← 1000 (3)

Do $\gamma \leftarrow 1$ and $k \leftarrow 1$ (4)

FOR k = 1 to MaxIter (5)

Solve LD-Sub1 and LD-Sub2 with $\hat{\mathbf{u}}^k$ (6)

Do $\mathrm{OFV}^k \leftarrow Z_{LD1}(\mathbf{x}^k, \hat{\mathbf{u}}^k) - Z_{LD2}(\mathbf{y}^k, \hat{\mathbf{u}}^k)$ (7)

IF $\mathrm{OFV}^k >$ BKLB (8)

Do BKLB ← OFV^k and $N^{\mathrm{UI}} \leftarrow 0$ (9)

ELSE (10)

Do $N^{\mathrm{UI}} \leftarrow N^{\mathrm{UI}} + 1$ (11)

END IF (12)

Do

$$\mathbf{s}^k \leftarrow \mathbf{x}^k - \mathbf{y}^k \tag{13}$$

Do

$$\delta_k \leftarrow \begin{cases} -\tau \dfrac{\mathbf{s}^k \boldsymbol{\Delta}^{k-1}}{\|\mathbf{s}^k\|^2}, & \text{if } \mathbf{s}^k \boldsymbol{\Delta}^{k-1} < 0 \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

Do

$$\boldsymbol{\Delta}^k \leftarrow \mathbf{s}^k + \delta_k \boldsymbol{\Delta}^{k-1} \tag{15}$$

Do

$$\lambda_k = \gamma \frac{\text{BestUB-OFV}^k}{\|\mathbf{s}^k\|^2} \tag{16}$$

IF $N^{\mathrm{UI}} = N^{\mathrm{MaxUI}}$ **Do** $\gamma \leftarrow \gamma *$DecRate and $N^{\mathrm{UI}} = 0$ (17)

IF $\mathbf{x}^k = \mathbf{y}^k$ (18)

Break FOR with optimal solution $\mathbf{x}^k$ (19)

ELSE IF $\left|\mathrm{OFV}^k - \mathrm{OFV}^{k-1}\right| < 0.0001$ (20)

Break FOR (21)

ELSE IF $(\text{BKUB-BKLB})/\text{BKLB} < 0.001$ (22)

Break FOR (23)

ELSE IF $\gamma < 0.00001$ (24)

Break FOR (25)

END IF (26)

Do $k \leftarrow k + 1$ (27)

END FOR (28)

Figure 2.7: Pseudo-code of the hybrid sub-gradient algorithm

5. Numerical Experiments

In studying the BSMPwRuDD, we wanted to answer several questions. First, can assigning a lot to different row depths be justified economically and, if so, under what conditions should relocation be pursued? Second, if relocation should be performed, is the solution procedure developed for DBSP efficient and reliable? To answer these questions, we performed a number of numerical experiments. All experiments were conducted on Intel Xeon Processor X5670 (hexa-core, 12M cache, 2.93 GHz) with 24 GB RAM and execution files run UNIX platform.

For numerical experiments, we randomly generate instances of three groups, as defined in Table 2.6. For details of the random generation, refer to Appendix B of this paper. Each group is characterized by the set of the number of lots, the set of the number of row depth types, and the set of the time horizons. Based on the number of variables, we refer to the instances in Group 1, Group 2, and Group 3 as small-sized, medium-sized, and large-sized problems. We consider a medium-sized problem to be a practical-sized problem. However, we are aware that large-sized problems exist, but not as commonly as medium-sized problems.

Table 2.6: Summary of instances randomly generated

		Group 1	Group 2	Group 3
Set of the number of lots		{10, 15, 20}	{30, 40, 50}	{100, 150, 200}
Set of the number of row depth types		{4, 5, 6}	{6, 7, 8}	{8}
Set of the time horizon		{20, 30, 40}	{30, 60, 90}	{180}
Number of instance types		27	27	3
Number of instances		135	135	15
Number of row positions in storage area	Average	16.33	32.32	35.07
	Min	6	22	31
	Max	30	43	41
Number of decision variables	Average	11,560	119,212	1,728,016
	Min	3,208	32,412	1,152,016
	Max	28,812	288,016	2,304,016

A single instance type is defined by mixing elements of three sets. To specify the instance type, we use the three-tuple ($L|R|T$) where L, R, and T respectively indicate the number of lots, the number of row depth types, and the time horizon. Taking all combinations of the possible values of each element, 27, 27, and 3 instance types are defined in Group 1, 2, and 3 respectively. Then, we create five cases per instance type, yielding a total of 135, 135, and 15 instances for Group 1, 2 and 3. For each instance type, the number of decision variables of the corresponding IP-BSMPwRuDD is computed by $LR^2T + 2R$.

Based on the number of decision variables, we determine the size factor of each instance type. It is computed by dividing the number of decision variables of each instance type by the number of decision variable of the ($L|R|T$)-instance type, which has the minimum number of decision variables among the instance types considered. For example, the number of decision variables of (10|4|20)-instance type is 3,208 and its size factor is 1. The number of decision variables of (2|6|40)-instance type is 28,812 and its size factor is 8.98. The number of decision variables of (200|8|90)-instance type, the largest instance type, is 1,152,016 and its size factor is 359.11.

Group 1 is designed to generate instances optimally solvable by CPLEX within the computation time limit. Therefore, the optimization problems of Group 1's instances have relatively few decision variables. According to the outcome of the CPLEX algorithm, we categorize instances into three classes: optimal instances, feasible instances, and unsolved instances. These classes respectively include the instances in which CPLEX finds an optimal solution within the computation time limit, CPLEX finds a feasible solution but is unable to conclude its optimality, and CPLEX fails to identify a feasible solution. When the given computation time limit is ct, we refer to them as OI(ct), FI(ct), and UI(ct).

Table 2.7: CPLEX outcome of Group 1 instances after three hours

# of lots		10			15			20			Total
Horizon		20	30	40	20	30	40	20	30	40	
# of depth type	4	5|0|0	5|0|0	5|0|0	5|0|0	5|0|0	5|0|0	5|0|0	5|0|0	5|0|0	45|0|0
	5	5|0|0	5|0|0	2|3|0	2|3|0	1|4|0	0|5|0	0|5|0	0|4|1	0|5|0	15|29|1
	6	5|0|0	5|0|0	5|0|0	0|3|2	0|5|0	0|5|0	0|3|2	0|3|2	0|4|1	15|23|7
Total		15|0|0	15|0|0	12|3|0	7|6|2	6|9|0	5|10|0	5|8|2	5|7|3	5|9|1	75|52|8
		42|3|0			18|25|2			15|24|6			

Table 2.7 provides the CPLEX outcome of Group 1's 135 instances as three-tuples in which the first, second, and third elements indicate the number of OI(3), FI(3), and UI(3). Within the three-hour computation time limit, 75 instances reach optimality, 52 additional instances are concluded feasible, and no feasible solution is identified for the remaining 8 instances. As the number of product lots, the number of row depth types, and the time horizon increases, the number of OI(3) tends to decrease; the number of FI(3) and UI(3) tends to increase.

Group 2 generates instances of sufficiently large-size to provide meaningful data for analysis of changeable row depth and relocation behavior. Within three hours of computation time, CPLEX identifies a feasible solution (but fails to conclude optimality) for 14 instances and fails to identify a feasible solution for the remaining 121 instances.

Group 3 is designed to generate instances of unusual size. Thus, the optimization problem corresponding to the instances has a very large number of decision variables, over one million. With the computation time limit of six hours, all 15 instances of Group 3 are UI(6).

In this section, we analyze the results of numerical experiments, providing insight into block stacking multiple products with relocation under deterministic demand. At first, we validate DH in Section 5.1 by benchmarking it against solving IP-BSMPwRuDD using the CPLEX 12.6.3 branch and cut algorithm. Section 5.2 verifies the benefit of changing row depths in block

stacking operations and Section 5.3 investigates the relocation behavior of the block stacking storage system, in the day, and of the product lot under different space utilization levels.

5.1. Validation of DH

In this section, we benchmark DH against CPLEX 12.6.3. At first, we compare the Objective Function Value (OFV) of the feasible solutions obtained by DH and CPLEX to check the reliability of DH. Next, we analyze lower bounds obtained by CPELX and DH. It shows the performance of the CPLEX and LD in generating the lower bound by benchmarking them against the linear relaxation of PreMP (LR). At the end, we study the optimality gap and the computation time of DH. We evaluate the applicability of DH in solving practical-sized instances and its resistance to the problem size. For convenience, let $f^*, \bar{f},$ and $\underline{f}$ be an optimal OFV, an upper bound on f^*, and a lower bound on f^*. We use the superscription of C, D, and L to indicate solution procedure of CPLEX, DH, and LR-PreMP. A control parameter of the THD heuristics ω is set as three based on the results of preliminary experiments.

5.1.1. Reliability of feasible solutions

In this section, the results of solving instances using CPLEX and DH are compared. For the analysis, we consider the instances of Group 1. Table 2.8 summarizes the results of the comparison between CPLEX and DH. The third column represents the gap between the OFVs of the solutions obtained by CPLEX and DH. It is computed by $\left(\bar{f}^D\text{-}f^*\right)/f^*$ for OI(3)s and $\left(\bar{f}^D\text{-}\bar{f}^C\right)/\bar{f}^C$ for FI(3)s. The fourth and fifth column show the average computation time of CPLEX and DH. For FI(3)s, we consider the computation time limit of three hours as CPLEX's computation time.

Table 2.8: Comparison of CPLEX and DH using data collected from instances of Group1

	Number of instances	Ave size factor	Gap between OFVs(%)	Ave. Computation Time (Sec)	
				CPLEX	DH
OI(3)	75	2.47	1.07	734.43	12.01
FI(3)	52	4.96	0.61	10800	26.64
UI(3)	8	5.36	-	10800	31.11

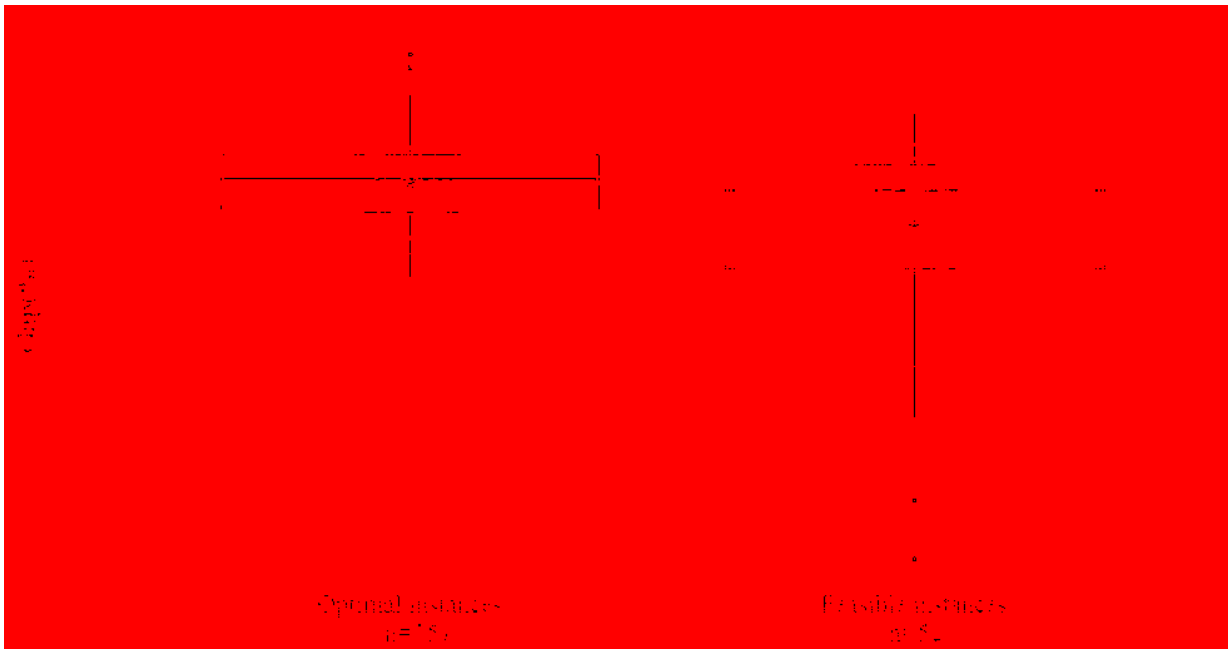

Figure 2.8: Comparison of OFVs of solutions obtained by CPLEX and DH

Figure 2.8 compares the OFVs obtained by CPLEX and DH by combining a box-and-whisker plot and a jittering-scatter plot. Concerning the box-and-whisker plot, the line in the box represents the median; the lower and the upper boundary of the box indicate the first quartile and the third quartile. The lower and the upper end of the vertical line emanating from the box correspond to the minimum and the maximum excluding outliers. The solid dots on the extension of the vertical line express the outliers. The "*" point represents the average. The jittering-scatter plot is a scatter plot where the random noise is added to data points to alleviate overlapping among them in the graph. In Figure 2.10, a jittering-scatter plot is represented by the set of

transparent dots. Each transparent dot corresponds to a single data point (a gap computed from an instance) and the random noise is added to its x-axis value (instance type), maintaining the original y-axis value (gap). The scattering shape of data points expresses a distribution along the vertical axis in the group of OI(3)s and FI(3)s.

Summarizing Table 2.8 and Figure 2.8, the average gap is 1.0%; the range is from 0% to 2.4%, including outliers in the case of OI(3)s; the average gap is 0.61% and the range is from -3.0% to 1.8%, including outliers in the case of FI(3)s. The graph of FI(3)s in Figure 2.8, includes 11 transparent dots located in the area of the negative gap; therefore, for 11 FI(3)s among 52 FI(3)s, or 21.15% of FI(3)s, DH determines a better feasible solution compared to solutions obtained by CPLEX.

Noting the small average and narrow range of the optimality gaps, we concluded the quality of the solution obtained by DH is acceptable compared to CPLEX for small-sized instances.

For the optimal instances, CPLEX's average computation time is 734.43 seconds, and the standard deviation is 1,903.25 second; DH's corresponding average and standard deviation are 12.01 seconds and 4.43 seconds. Among 75 OI(3)s, CPLEX is faster for 38 instances and DH is faster for 37 instances; however, whereas CPLEX is faster by 8.41 seconds on average (in the 38 instances), DH is on average faster by 1,473.00 seconds (in the 37 instances).

5.1.2. Quality of lower bound

In this section the lower bound of CPLEX, DH, and LR are compared. Because CPLEX provides a lower bound even for UI(3), we consider the results of all instances for the analysis. Table 2.9 summarizes the comparison of the lower bounds.

The average gap column represents how far the lower bound obtained by CPLEX, DH, and LR is from the best-known upper bound. Let $\bar{f}^{\text{best}}$ be the best-known upper bound determined

by min($\bar{f}^C$, $\bar{f}^D$). Let gap^C, gap^D, and gap^L be the gap of CPLEX, DH, and LR, computed by $\left(\bar{f}^{\text{best}} - \underline{f}^C\right)/\underline{f}^C$, $\left(\bar{f}^{\text{best}} - \underline{f}^D\right)/\underline{f}^D$, and $\left(\bar{f}^{\text{best}} - \underline{f}^L\right)/\underline{f}^L$, respectively.

The average difference column represents the difference between gap^C and gap^L computed by gap^L-gap^C and between gap^D and gap^L calculated by gap^L-gap^D. The average difference decreases as the instance size factor increases and is negligible in for medium-sized and large-sized instances. Based on this observation and computation time, LR is an attractive alternative for establishing a good lower bound quickly for medium-sized and large-sized instances. Notice, the lower-bounding procedure of DH consists of LD heuristics and LR. If LD heuristics are omitted in solving large-sized instances, on average, the total computation time of DH is reduced by 1,499.64 seconds or 21.76%.

It is known Lagrangean decomposition generally provides a better lower bound compared to the linear relaxation (Guignard and Kim, 1987). The average difference column in Table 2.9 shows the difference between the lower bounds established by LD heuristics and LR is not significant. Two possible reasons for the deterioration of the performance of LD heuristics are the number of decision variables and the excellence of the lower bound obtained by LR. The first reason is supported by a logical assumption: if the original problem has so many decision variables, finding an optimal Lagrangean multiplier is as difficult as finding an optimal solution of the original problem. The second reason is based on an observation from Table 2.9: in the case of OI(3)s of Group 1, the average gap is 2.70%, meaning the lower bound obtained by LR is very close to an optimal solution.

Table 2.9: Comparison of the lower bounds obtained by CPLEX, LD, and LP relaxation

# of lots		# of instances	Ave size factor (ASF)	Ave gap (%)			Ave difference (% points)		Ave computation time (sec)		
				C	D	L	C	D	C	D	L
Group 1	10	45	2.40	0.06	3.18	4.38	4.32	1.20	1732.50	11.42	0.19
	15	45	3.60	1.51	2.47	2.89	1.39	0.43	6212.65	18.74	0.29
	20	45	4.80	1.68	2.16	2.36	0.68	0.20	5762.14	26.18	0.38
Group 2	30	45	27.87	4.20	4.36	4.48	0.28	0.12	10800.00	175.84	3.61
	40	45	37.16	3.60	3.69	3.75	0.15	0.06	10800.00	246.13	5.72
	50	45	46.45	3.49	3.55	3.59	0.10	0.04	10800.00	315.80	6.88
Group 3	100	5	179.56	4.40	4.42	4.44	0.03	0.01	43200.00	4304.01	638.22
	150	5	269.33	4.27	4.29	4.29	0.02	0.00	43200.00	7711.91	1050.59
	200	5	359.11	2.72	2.73	2.73	0.01	0.00	43200.00	11868.91	1472.65

5.1.3. Duality gap and computation time of DH

This section addresses the change in the optimality gap and the computation time of DH as instance size increases. The optimality gap is computed by $\left(\bar{f}^D - \underline{f}^D\right)/\underline{f}^D$. Thus, the gap between $\bar{f}^D$ and unknown f^* is less than the reported optimality gap. Table 2.10 summarizes the average optimality gap and average computation time, showing the optimality gap remains at a specific range around 3.5% regardless of the instance size. The computation time naturally increases as the instance size increases.

Table 2.10: Performance of the DH

Number of lots		Number of instances	Ave size factor (ASF)	Ave duality gap (%)	Ave computation time (sec)
Group 1	10	45	2.40	4.27	11.42
	15	45	3.60	3.45	18.74
	20	45	4.80	2.85	26.18
Group 2	30	45	27.87	4.36	175.84
	40	45	37.16	3.72	246.13
	50	45	46.45	3.58	315.80
Group 3	100	5	179.56	4.42	4304.01
	150	5	269.33	4.29	7711.91
	200	5	359.11	2.73	11868.91

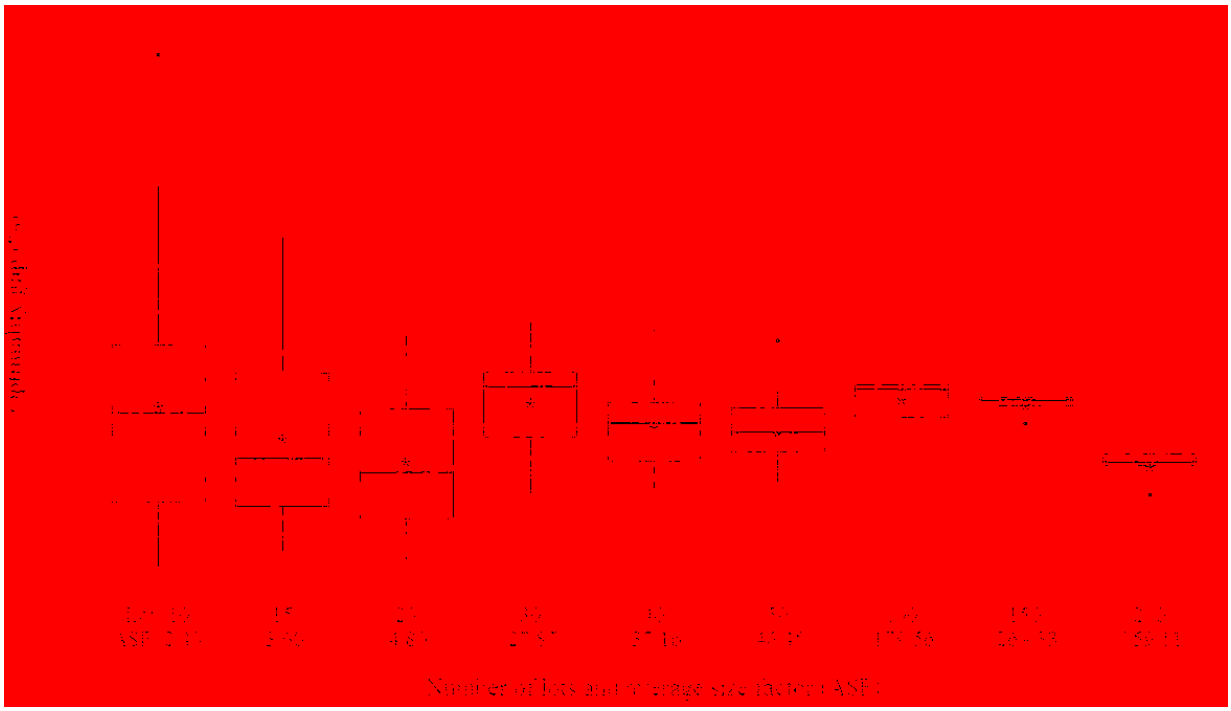

Figure 2.9: The change in the optimality gap as instance size increases

Using a combined box-and-whisker plot and jittering-scatter plot, Figure 2.9 illustrates the distribution of the instance's duality gap in the group of instances specified by the number of product lots. The "*" points represent the average gap within each group; the horizontal bar traversing the graph indicates the average gap (3.71%) across all 285 instances. In cases where the number of product lots is greater than 20, all observations are positioned in $\pm$3% area from the average line, excluding some outliers. The optimality gap does not seem to increase as a result of increasing the number of lots or average size factor.

To summarize the performance of DH based on the result of the experiment, it can determine a good feasible solution in a reasonable time regardless of instance size. Compared to CPLEX, this merit is highlighted when solving practical-sized instances.

5.2. Changeable row depth

This section investigates the benefit of dynamic block stacking compared to static block stacking. Additionally, we consider semi-dynamic block stacking in the comparison. From the viewpoint of changeable row depths, operational strategies can be defined as follows:

- Dynamic Block Stacking (DBS) allows changing the row depth designated for the product lot at any time during its storage life.
- Semi-Dynamic Block Stacking (SDBS) allows changing the row depth designated for the product lot only at the replenishment point.
- Static Block Stacking (SBS): The row depth designated for the product lot is permanent and changing row depth is not allowed.

SDBS is an interesting strategy of block stacking operation. Like DBS, SDBS is also not considered in the conventional literature of block stacking. It restricts changing row depths to replenishment points and doesn't allow relocation to change row depths. Compared to DBS, like SBS, it causes no additional material handling.

In the numerical experiments adopting DBS strategy in a block stacking operation, we observed cases where a product lot changes row depth. Table 2.11 shows the percentage of the cases where a product lot changes row depth with relocation and the percentage of the cases where a product lot changes row depth at the replenishment point with no relocation among all the cases. The fourth column and the fifth column represent on average over instances of all groups, the first one is 65.85% and the second one is 34.15%, respectively. It represents the percentage of the cases where a product lot changes row depth at the replenishment point is not low. Based on this observation, SDBS has a potential to be an attractive alternative of taking both DBS's advantage of reducing honeycomb loss and SBS's advantage of no additional material handling.

Table 2.11: Percentage of changing row depth by relocation and at replenishment point

	Number of instances	Ave size factor	Percentage of changing row depth with relocation	Percentage of changing row depth at replenishment
Group 1	135	3.60	65.87	34.13
Group 2	135	37.16	64.49	35.51
Group 3	15	538.66	77.98	22.02
Total	285	47.66	65.85	34.15

For the comparison, the optimization problems of SDBS and SBS are developed by modifying IP-BSMPwRuDD. In developing the optimization problem of SBS, the following constraint (2.57) is added to IP-BSMPwRuDD. It guarantees only one storage area is assigned for a product lot over a planning horizon and it cannot be changed.

$$\sum_{(i,tR+r)\in\overline{A}_{tR+r}} x^l_{i,tR+r} = \sum_{(k,(t+1)R+r)\in\overline{A}_{(t+1)R+r}} x^l_{k,(t+1)R+r}, \quad t=1,\dots,T \text{ and } r=1,\dots,R \tag{2.57}$$

In the optimization problem of SDBS, the following constraint (2.58) is added to IP-BSMPwRuDD. It requires only one storage area is assigned for a product lot during its inventory cycle and it can be changed at the replenishment point.

$$\sum_{(i,tR+r)\in\overline{A}_{tR+r}} x^l_{i,tR+r} = \sum_{(k,(t+1)R+r)\in\overline{A}_{(t+1)R+r}} x^l_{k,(t+1)R+r}, \quad t=1,\dots,T \text{ and } t \text{ is not a replenishment point and } r=1,\dots,R \tag{2.58}$$

In this section, we refer to the optimization problem of DBS, SDBS, and SBS as OP-DBS, OP-SDBS, and OP-SBS, respectively.

In the comparison, we use the results of OI(3)s and FI(3)s of Group 1 considering the following issue. Let $f^*_{(\cdot)}$ and $\bar{f}_{(\cdot)}$ be an optimal objective function value of the optimization problem assuming the strategy $(\cdot)$ and the upper bound on $f^*_{(\cdot)}$ computed by the best known feasible solution of the strategy $(\cdot)$. Theoretically, among the strategies, the relation of $f^*_{DBS} \leq f^*_{SDBS} \leq f^*_{SBS}$ is satisfied. When the analysis is based on the results of Group 2's instances, $\bar{f}_{DBS}$

is considered in the comparison instead of f^*_{DBS} because all instances of Group 2 are not the optimal instance for DBS. In the comparison using the instances of Group 2, many cases of $\bar{f}_{DBS} > f^*_{SDBS}$ and $\bar{f}_{DBS} > f^*_{SBS}$ were observed. It means if the comparison is based on the results of Group 2's instances, there is a strong possibility the relation of $f^*_{DBS} \leq f^*_{SDBS} \leq f^*_{SBS}$ is distorted and consequently, the benefit of DBS is underestimated. Therefore, we used OI(3)s and FI(3)s of Group 1 for the comparison. The results of these instances satisfy the relation of $f^*_{DBS} \leq f^*_{SDBS} \leq f^*_{SBS}$ or $\bar{f}_{DBS} \leq f^*_{SDBS} \leq f^*_{SBS}$.

In the remaining part of this section, we investigate each strategy's minimum storage capacity requirement and operating cost and compare the strategies under different size of storage spaces and different unit costs.

Comparison of the minimum storage space requirements

The Minimum Storage Space Requirement, or MSSR, represents the minimum storage space required to operate a block stacking system without using external storage space over a planning horizon, following general operational rules and the strategy about changing row depth. In the experiment, it is measured as the number of row positions in the storage area. We suppose all storage areas have the same number of row positions and consequently, it can reasonably quantify the storage space. We refer to a MSSR of each strategy as MSSR-DBS, MSSR-SDBS, and MSSR-SBS, respectively.

Table 2.12 summarizes the result of the experiment. On average, DBS, SDBS, and SBS require at least 16.39 row positions, 19.67 row positions, and 20.33 row positions, respectively. Compared to DBS, 20.02% and 24.04% more storage space is necessary for SDBS and SBS, respectively. Thus, DBS requires less storage space than SBDS and SBDS requires less storage space than SBS.

Table 2.12: The minimum storage space requirements for different operational strategies

	DBS	SDBS	SBS
Average minimum storage space requirement (MSSR) measured in the number of row positions	16.39	19.67	20.33
Ratio of each strategy's average MSSR to the DBS's average MSSR	1	1.2002	1.2404

Table 2.13: The daily operating costs of the different operational strategies when the storage space is set equal to the SBS's minimum storage capacity requirement

			DBS	SDBS	SBS
At the minimum storage space requirement of SBS	Average of the daily operating cost ($)	Total cost	3,497.25	3,528.37	3,559.64
		Space cost	2,732.90	2,782.14	2,818.29
		Material handling cost	764.35	746.22	741.35
	Average of the difference to the cost of DBS ($)	Total cost	-	31.11	62.38
		Space cost	-	49.24	85.39
		Material handling cost	-	-18.13	-23.00
	Average of the ratio to the cost of DBS	Total cost	-	1.0318	1.0409
		Space cost	-	1.0532	1.0654
		Material handling cost	-	0.9016	0.8970

Comparison of operating costs

In the experiment, when comparing the operating costs of DBS, SDBS, and SBS, the number of row positions in the storage area of OP-DBS, OP-SDBS, and OP-SBS are set equal to MSSR-SBS. Let OP-*X*(*Y*) be the optimization problem of the strategy *X* when there are *Y* row positions in each storage area. For example, if MSSR-SBS is 20 for an instance, we solve OP-DBS(20), OP-SDBS(20), and OP-SBS(20) and then compare their objective function values. Notice, MSSR-DBS is less than MSSR-SDBS and MSSR-SBS and, thus, in many cases, OP-SDBS(MSSR-DBS) and OP-SBS(MSSR-DBS) are infeasible and have no feasible solution.

Table 2.13 provides the results of the experiment. According to the second row, on average, the daily operating cost of DBS, SDBS, and SBS is about $3,497, $3,528, and $3,559. The fifth row shows the average of the difference between f^*_{DBS}(or $\bar{f}_{DBS}$) and f^*_{SDBS} and f^*_{DBS}(or $\bar{f}_{DBS}$) and f^*_{SBS}. On average, DBS saves about $31 per day and $62 per day compared to SDBS and SBS,

respectively. The eighth row represents the average of the ratio of f^*_{SDBS} and f^*_{SBS} to f^*_{DBS}(or $\bar{f}_{DBS}$). On average, SDBS and SBS result in 3.18% and 4.09% more operating cost than DBS.

The rows of total cost, space cost, and material handling cost show DBS incurs lower space cost and higher material handling cost compared to SDBS and SBS. The result implies the savings in space cost by changing row depth is greater than the increase in material handling cost by changing row depths; consequently, DBS decrease overall operating cost.

Comparison of the operating costs as storage space changes

Next, we investigate how the differences among the operating costs of DBS, SDBS, and SBS change as storage space increases. In the experiment for each instance, initially, OP-DBS(MSSR-DBS), OP-SDBS(MSSR-DBS), and OP-SBS(MSSR-DBS) are solved and their objective function values are compared. Then, we solve OP-DBS(MSSR-DBS+1), OP-SDBS(MSSR-DBS+1), and OP-SBS(MSSR-DBS+1) and compare their objective function values. Next, we solve OP-DBS(MSSR-DBS+2), OP-SDBS(MSSR-DBS+2), and OP-SBS(MSSR-DBS+2). This procedure is repeated, increasing the storage space parameter by 1 and stopping after solving OP-DBS(2*MSSR-DBS), OP-SDBS(2*MSSR-DBS), and OP-SBS(2*MSSR-DBS).

Figure 2.10 illustrates the change in the differences among the operating costs of DBS, SDBS, and SBS. Each graph depicts the change in the instances of 4 row depths, 5 row depths, and 6 row depths. In the graphs, storage space is represented by the ratio to MSSR-DBS. For example, if MSSR-DBS is 10 row positions, the space of 12 row positions is expressed as 1.2. Because we increase storage space by twice the value of MSSR-DBS, it ranges from one to two. The differences among the operating costs are measured as the ratio of $f^*_{\text{OP-SDBS}(\cdot)}$ and $f^*_{\text{OP-SBS}(\cdot)}$ to $f^*_{\text{OP-DBS}(\cdot)}\left(\text{or } \bar{f}_{\text{OP-DBS}(\cdot)}\right)$. For example, if the ratio of SDBS is 1.0425, it means SDBS requires 4.25% more operating cost compared to DBS for a given amount of storage space.

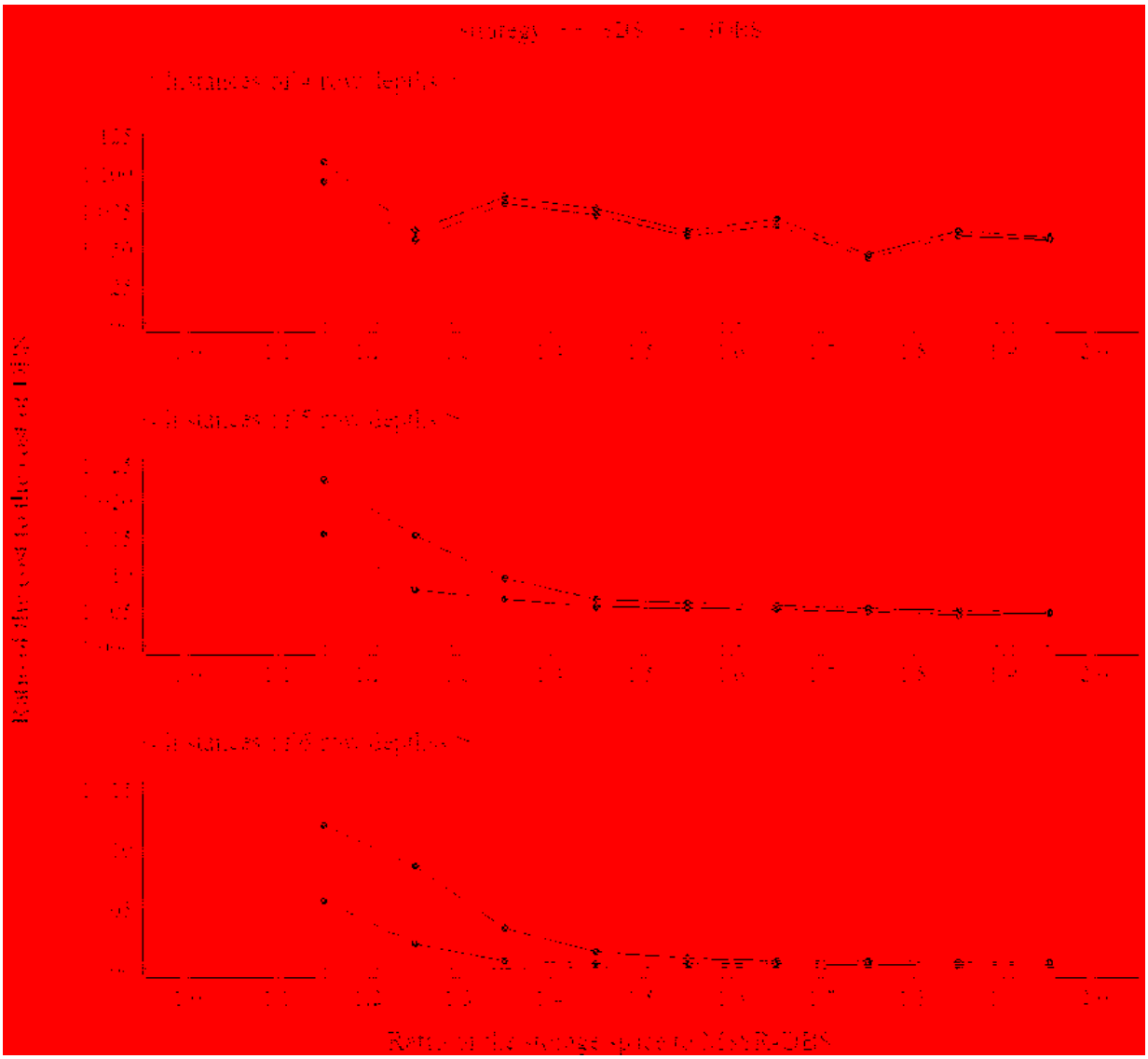

Figure 2.10: The ratio among the daily operating costs of the different operational strategies as storage capacity changes

To smooth the graph, the observations are classified into 10 groups of the storage space ratio and then, represented by the average of each group. The groups are defined by dividing the space ratio's range into ten groups equally. For example, the first group and the eighth group correspond to the ratio range from 1 to 1.1 and from 1.7 to 1.8, respectively. Notice, OP-SDBSs and OP-SBSs with the storage space parameters corresponding to the first group are infeasible problems and, thus, no observations about SDBS and SBS are in the first group.

In Figure 2.10, solid dots indicate the average ratio of observations of SDBS and SBS in each group; the lines pass through these points. All graphs show the lines of SDBS and SBS converge to the thick line representing the ratio of one as the space ratio increases. Thus, the gaps among the operating costs of DBS, SDBS, and SBS decrease as storage space increases. Considering inventory level is fixed as storage space increases, the result implies, compared to SDBS and SBS, DBS performs well when the ratio of inventory level to storage space is low.

Comparison of operating costs under different unit costs

In this subsection, we analyze the differences among the operating costs of DBS, SDBS, and SBS with different floor space and material handling unit costs. Table 2.14 summarized the results.

Figure 2.11 shows the ratio of SDBS-to-DBS and SBS-to-DBS decreases as material handling unit cost increases. In addition, it illustrates the ratio of SDBS-to-DBS and SBS-to-DBS increases as floor space unit cost increases.

Table 2.14 and Figure 2.11 indicate within the range of floor space unit cost of $0.20/sqtf/day to $0.24/sqft/day and material handling unit cost of $0.40/min to $0.48/min, DBS incurs less operating cost compared to SDBS and SBS.

Table 2.14: Comparison of the daily operating costs of the different operational strategies when different unit costs are assumed and storage capacity is set as the SBS's minimum storage capacity requirement

Unit cost of FS	Unit cost of MH	Average daily operating cost at the minimum required storage space of SBS ($)			Ave ratio to the DBS's operating cost		
		DBS	SDBS	SBS	DBS	SDBS	SBS
0.20	0.40	3,179.32	3,207.55	3,236.05	1	1.0318	1.0409
0.20	0.48	3,317.45	3,342.86	3,370.69	1	1.0286	1.0372
0.22	0.44	3,497.25	3,528.37	3,559.64	1	1.0318	1.0409
0.24	0.40	3,675.42	3,713.17	3,748.29	1	1.0352	1.0448
0.24	0.48	3,815.12	3,849.02	3,883.21	1	1.0318	1.0409

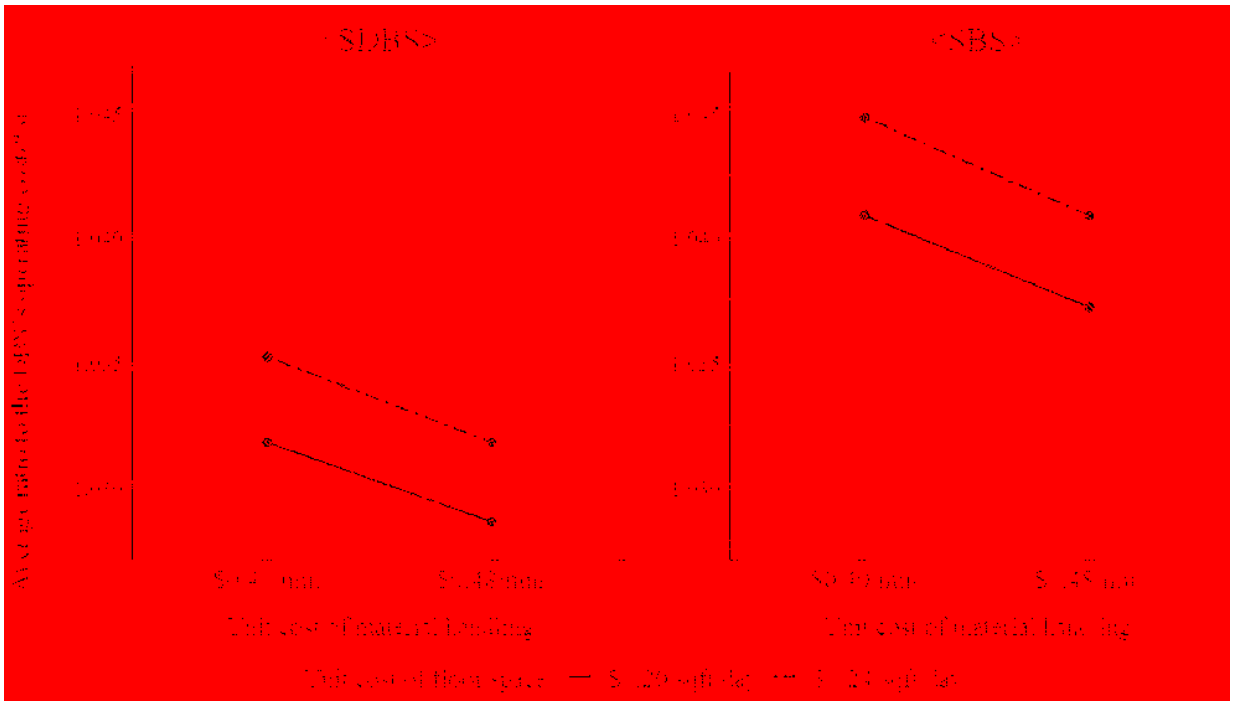

Figure 2.11: The ratio of the operating cost of SDBS and SBS to the cost of DBS as unit cost of material handling changes

An interesting observation in Table 2.14 is the average ratio of SDBS's cost and SBS's cost to DBS's cost are almost the same when material handling unit cost is twice the floor space unit cost: $0.20/sqft/day and $0.40/min; $0.22/sqft/day and $0.44/m; $0.24/sqft/day and $0.48/min. Although it appears a linear relationship exists among floor space unit cost, material handling unit cost, and the ratio of the operating cost, the result of a linear regression analysis indicates there is no stochastically significant linear relationship among them.

To summarize, DBS outperforms SDBS and SBS based on the experiments' results of Section 5.2. Especially, DBS's minimum storage space requirement is 15.38% less than compared to SDBS and 15.97% compared to SBS. It implies given the same size of the storage space, DBS can provide more storage capacity compared to SDBS and SBS. Thus, if storage space is insufficient in operating a block stacking storage system adopting conventional SBS, applying DBS instead of SBS should be considered expanding storage space.

In addition, DBS incurs less operating cost compared to SDBS and SBS when storage space is not sufficient considering the inventory level. When the size of the storage space equals to the SBS's minimum storage space requirement, SBS incurs 4.09% more operating cost compared to DBS. However, difference between operating cost of DBS and SBS is insignificant when the space utilization is low. Thus, when space utilization is high, adopting DBS instead of SBS should be considered in order to reduce operating cost.

Notice, given a feasible instance of DBS, an optimal solution of SBS outperforms the feasible solution of DBS in many cases. Thus, a manager should be careful in making decisions between DBS and SBS. If an optimal solution of SDBS is possible, SDBS would be the best alternative compared to a feasible solution of DBS and an optimal solution of SBS.

5.3. Relocation behavior

In this section, we investigate relocation behavior at the storage system level and at the product lot level. Based on an analysis of the results from the experiments, Section 5.3 provides insights regarding block stacking with relocation.

5.3.1. Relocation behavior of the storage system

In this section, we quantify relocation behavior using the following three value measurements: average number of relocated product lots per day, average number of relocated unit loads per day, and average relocation cost per day. Additionally, to standardize these value measurements over different-sized instances, we use the following three ratios: the ratio of the average number of relocated product lots per day to the total number of product lots, the ratio of the average number of relocated unit loads per day to the average inventory level per day, and the ratio of the average relocation cost per day to the average total operating cost per day.

Table 2.15: Summary of relocation behavior of the storage system

Number of lots	Number of row depth types	Number of instances	Relocated lots per day		Relocated unit loads per day		Relocation cost per day	
			Lots	Ratio (%)	Unit loads	Ratio (%)	$	Ratio (%)
30	6	15	3.72	12.41	183.44	5.18	231.17	3.03
	7	15	3.97	13.22	370.61	6.48	515.52	4.22
	8	15	4.11	13.71	455.01	6.45	608.75	4.13
40	6	15	5.02	12.55	160.97	4.85	204.29	2.76
	7	15	4.86	12.15	303.21	5.35	421.69	3.40
	8	15	5.87	14.67	497.61	6.86	672.77	4.20
50	6	15	8.21	16.42	247.95	7.74	324.82	4.46
	7	15	7.45	14.91	382.26	6.99	540.39	4.39
	8	15	6.86	13.72	465.39	6.13	650.32	3.94

Table 2.15 summarizes the value measurements and the ratios. The results are organized according to the number of lots and the number of row depths considered in instances.

Relocation behavior of storage system as daily space utilization changes

To reorganize data, we use the concept of Stack Position Utilization, or SPU. It represents a kind of space utilization based on the number occupied stack positions and the number of total stack positions. The storage system's SPU at day t is computed by

$$(\#\text{ of stack positions occupied at day } t)/(\text{total } \#\text{ of stack positions}). \tag{2.59}$$

Considering the number of occupied stack positions is computed based on the inventory level, SPU represents the relative inventory level. For example, consider the inventory level of 1,000 unit loads or 450 stacks at day t. When given a storage space of 500 stack positions, the storage system's SPU is 0.9 and y the inventory level is relatively high at day t. When given a storage space of 900 stack positions, the storage system's SPU is 0.5 and the inventory level is relatively low at day t. Thus, SPU-values closer to one indicate relatively higher inventory levels.

Table 2.16: The percentage measurements of the number of relocated lots and the number of relocated unit loads at each day.

Range of SPU	Number of days	The average ratio of the number of relocated lots to the total number of product lots at the day (%)	The average ratio of the number of relocated unit loads to the inventory level at the day (%)
< 0.82	340	7.81	2.35
0.82 - 0.86	1465	8.92	3.19
0.86 - 0.90	2799	11.46	4.73
0.90 - 0.94	2759	15.08	6.91
≥ 0.94	737	25.27	14.17

Table 2.16 organizes the observations into five categories defined by the range of SPU. Totally, observations of 7,100 days are used in the analysis. The third column of the average ratio of the number of relocated lots to the total number of product lots at the day and the fourth column of the average ratio of the number of relocated unit loads to the inventory level at the day shows the ratios increase as SPU increases.

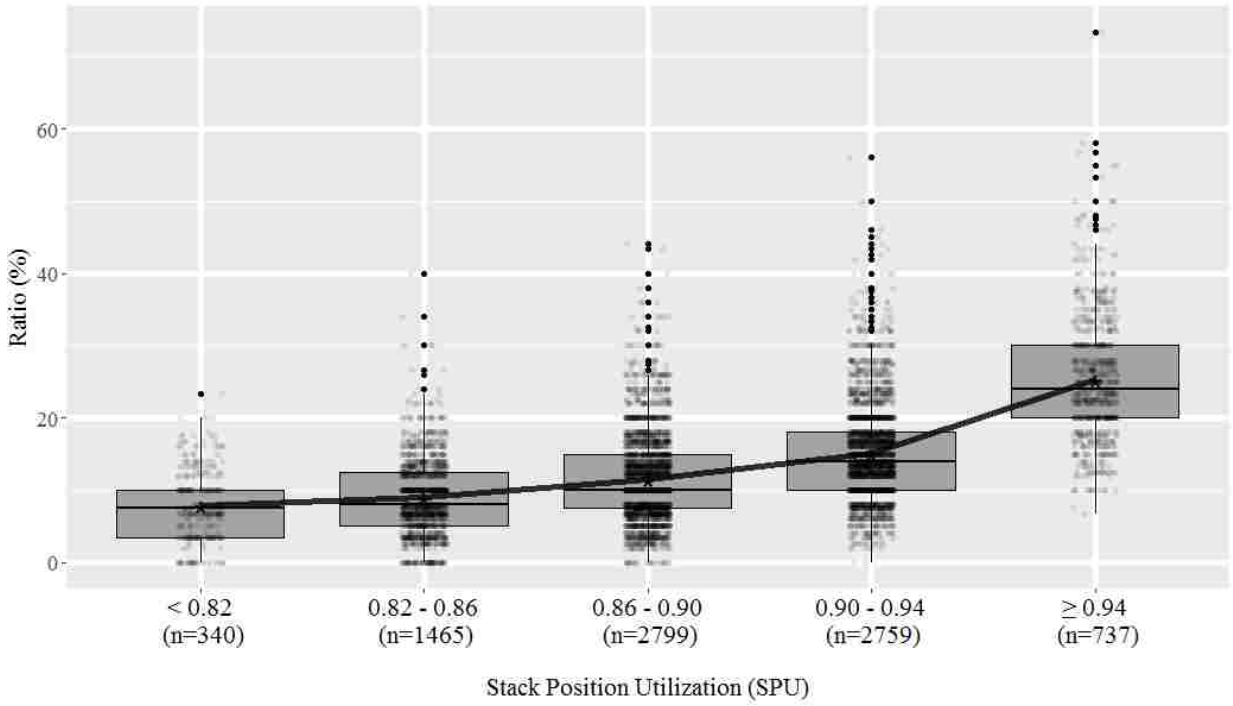

Figure 2.12: The average ratio of the number of relocated lots to the total number of product lots at the day as SPU changes

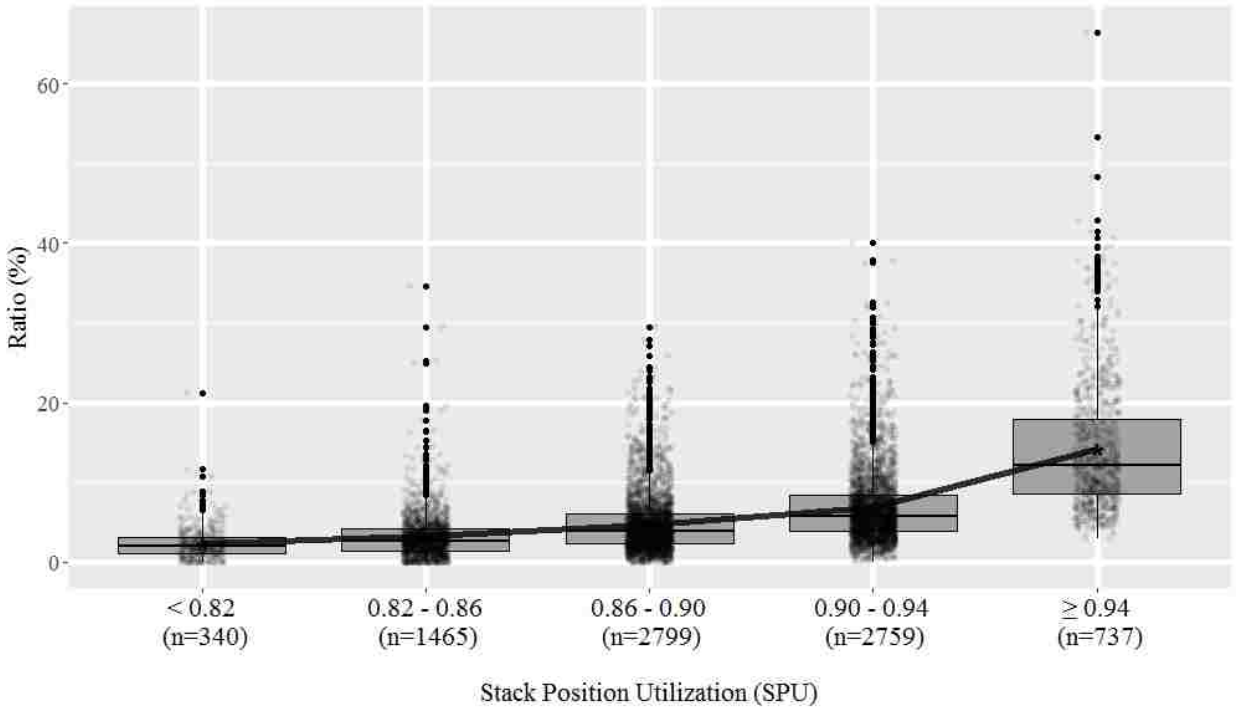

Figure 2.13: The average ratio of the number of relocated unit loads to the inventory level at the day as SPU changes

Figure 2.12 and Figure 2.13 illustrate the dispersion of the observations of the ratio of the number of relocated lots to the total number of product lots at the day and the ratio of the number of relocated unit loads to the inventory level at the day. Figure 2.12 and Figure 2.13 combine the box plot and the Jitter graph. The "*" point indicates the average value of the categories defined by the range of SPU; the solid line passes through these average points. These graphs show the increasing tendency of the ratios of the relocation behavior as the day's SPU increases.

To summarize, if the day's SPU is high, more relocations are expected.

Relocation behavior of the storage system as storage capacity changes

Figure 2.14 addresses how relocation behavior changes as storage capacity changes. For each instance, we conducted an experiment by comparing the relocation behavior measurements of OP-DBS(MSSR-DBS), OP-DBS(MSSR-DBS+1) , …, OP-DBS(2*MSSR-DBS).

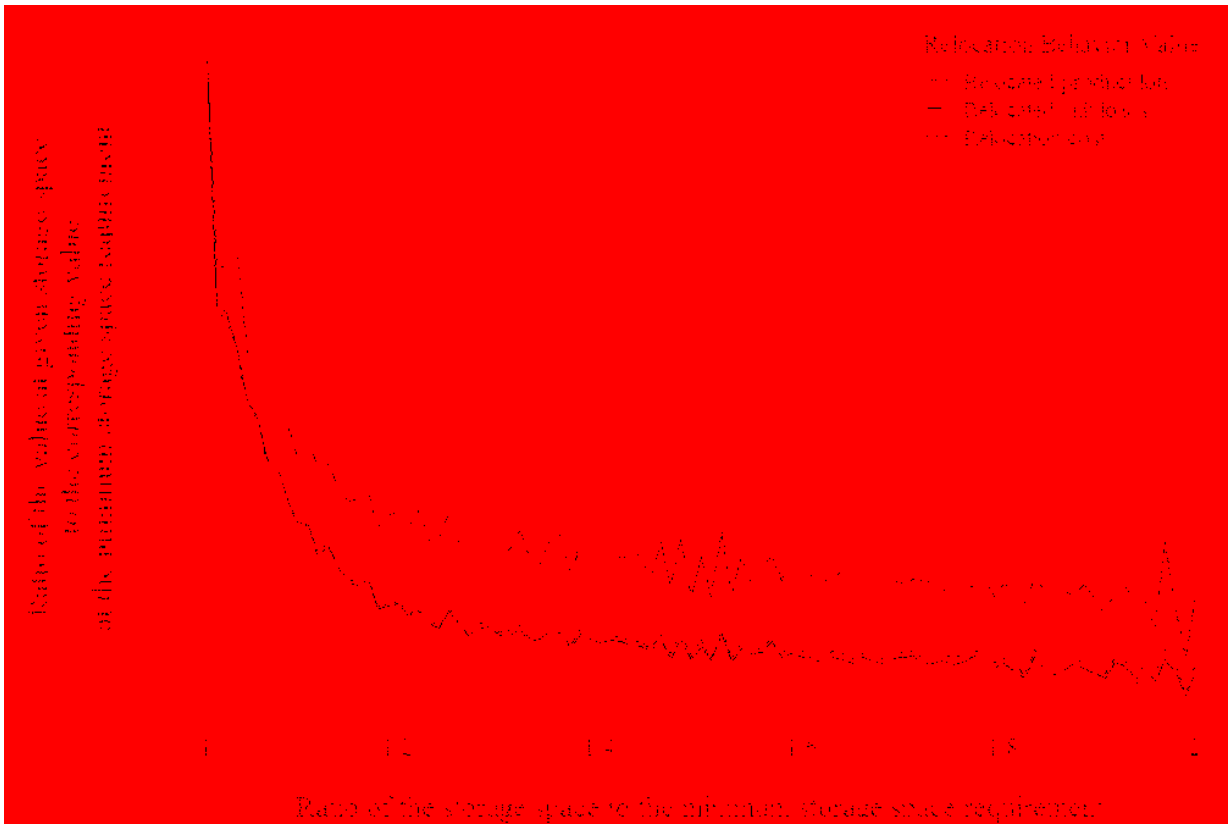

Figure 2.14: The percentage measurements of the relocation behavior as the storage capacity changes

In Figure 2.14, the relocation behavior change is quantified as the ratio of the percentage measurements at a given storage space to the percentage measurements at MSSP-DBS. Because relocation is most active at MSSP-DBS, the factors are ranged from one to zero. Storage space is expressed as the ratio to MSSP-DBS and, thus, the storage space factors are ranged from one to two. The closer to one on the *x*-axis, the higher storage system's SPU; the closer to two, the lower storage system's SPU.

The graph shows the ratio of the percentage measurements of relocation behavior decreases as the ratio of storage space increases. It infers a block stacking storage system of sufficient storage space (or the lower SPU) can anticipate fewer relocations compared to a block stacking storage system with insufficient storage space or higher SPU. Relocations tend to occur more frequently when the storage system's average space utilization or inventory level is relatively high.

Table 2.17: Comparison of relocation behaviors of the storage system under different cost parameters

Unit cost of FS	Unit cost of MH	Relocated lots per day		Relocated unit loads per day		Relocation cost per day	
		Lots	Ratio (%)	Unit loads	Ratio (%)	$	Ratio (%)
0.20	0.40	5.56	13.75	340.13	6.21	420.49	3.83
0.20	0.48	5.43	13.4	334.19	6.09	494.49	4.3
0.22	0.44	5.56	13.75	340.72	6.23	463.3	3.84
0.24	0.40	5.76	14.25	348.68	6.4	432.54	3.42
0.24	0.48	5.55	13.71	340.36	6.21	505.18	3.83

One interesting feature of Figure 2.14 is the similarity with regard to relocated unit loads and relocation cost. Considering relocation cost is proportional to the number of relocated unit loads, this is an expected result.

Relocation behavior of the storage system under different cost parameters

In this section, we analyze the relocation behavior under different floor space cost and material handling cost. In the experiment, for each instance, we solve OP-DBS(MSSR-DBS) with different unit cost factors and compare their OFVs.

Table 2.17 summarizes the result according the unit cost factors considered. Like Table 2.14, the percentage measurements are the almost same when the ratio of the unit floor space cost to the unit material handling cost is two. Likewise, the result of the linear regression analysis indicates there is no stochastically significant linear relationship among floor space unit cost, material handling unit cost, and the ratio of the operating cost.

Figure 2.15 illustrates the ratio of the average number of relocated product lots per day to the total number of product lots and the ratio of the average number of relocated unit loads per day to the average inventory level per day decreases and the ratio of the average relocation cost per day to the average total operating cost per day increases. Thus, when unit material handling cost is relatively high compared to unit floor space cost, relocation occurs less frequently because the

savings in the floor space cost by the changing row depth is relatively small compared to the increase in material handling cost resulting from relocation. The higher percentage of the relocation cost means lower percentage of the floor space cost in the total operating cost.

In addition, Figure 2.15 shows the percentage or the number of relocated lots and the percentage of the number of relocated unit loads increases and the percentage of the relocation cost decreases as unit floor space cost increases. Thus, when unit floor space cost is relatively high compared to unit material handling cost, the relocation occurs more frequently because the savings in the floor space cost by changing row depth is relatively large compared to the increase in material handling cost due to relocation. The lower percentage of the relocation cost means higher percentage of the floor space cost in the total operating cost.

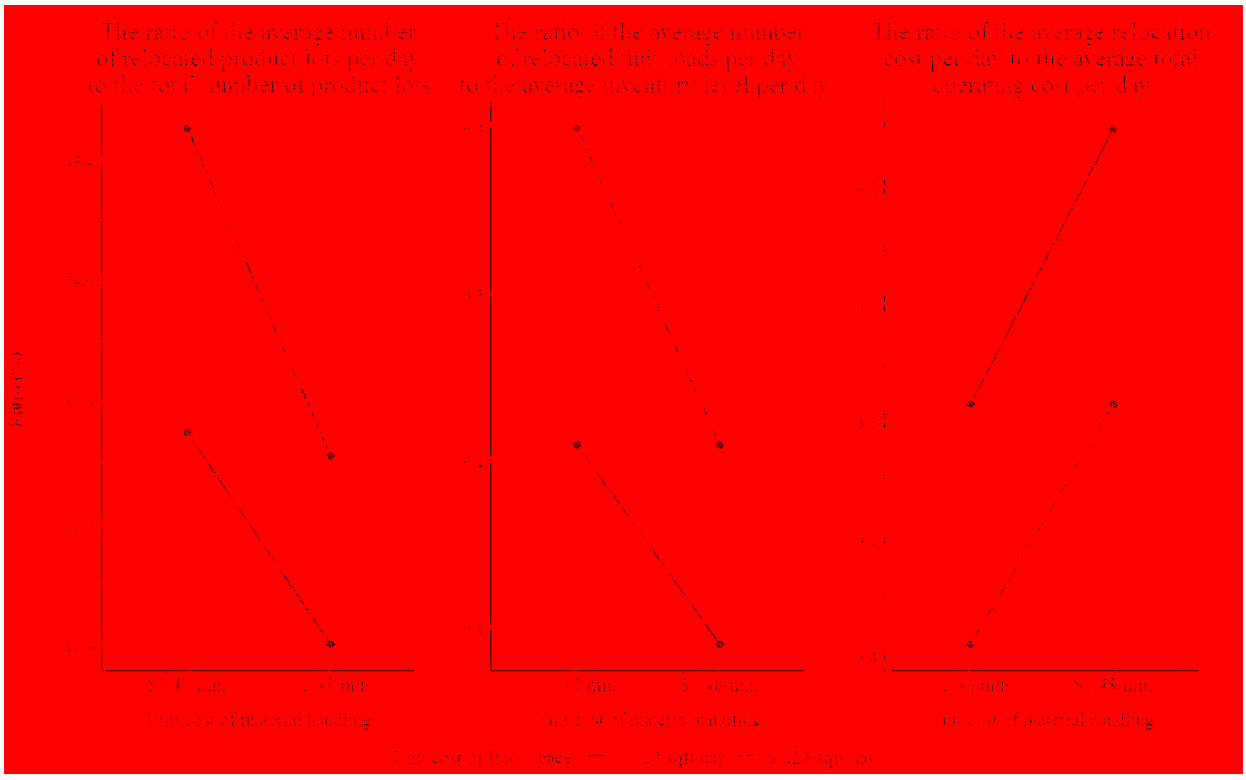

Figure 2.15: The ratio measurements of the relocation behavior as unit material handing cost changes

To summarize, based on the results of this subsection, relocation behavior is controlled by the relationship between floor space unit cost and material handling unit cost. Relocation behavior increases when floor space unit cost is relatively high, whereas it decreases when material handling unit cost is relatively high. Thus, relocation behavior is a function of unit costs and a well-designed DBS functions as a self-regulating strategy.

5.3.2. Relocation behavior of the product lot

In this section, we investigate the relocation behavior of the product lot. The data are grouped according to the ratio of the average stack level of the product lot to average stack level of the storage system: the Ratio of the Average Stack Level or RASL. RASL greater than one means a product lot's average inventory level is relatively high compared to other product lots. RASL less than one means the product lot's average inventory level is relatively low compared to other product lots. Totally, observations of 5,400 product lots are used in the analysis.

Relocation behavior of a product lot is quantified using two value measurements: the expected number of relocations of the product lot per day and the expected number of relocated unit loads of the product lot per day. Additionally, to standardize these value measurements over different-sized instances, we use the following two ratio measurements: the ratio of the expected number of relocations of the product lot per day to the expected number of relocations per day and the ratio of the expected number of relocated unit loads of the product lot per day to the expected number of relocated unit loads per day.

Table 2.18 organizes the observations of the value measurements and the ratio measurements of the relocation behavior of the product lot into eight categories defined by the range of RASL.

Table 2.18: Relocation summary based on lots inventory level

Range of RASL	Number of lots	Expected number of relocations of the product lot per day		Expected number of relocated unit loads of the product lot per day	
		Per day	Ratio (%)	Per day	Ratio (%)
< 0.4	616	0.1930	3.34	5.5933	0.47
0.4 - 0.6	801	0.1672	3.06	8.6405	0.84
0.6 - 0.8	854	0.1598	2.95	9.7706	1.13
0.8 - 1.0	780	0.1420	2.60	9.6656	1.47
1.0 - 1.2	670	0.1220	2.20	8.7039	1.72
1.2 - 1.4	554	0.1146	2.01	8.8016	1.80
1.4 - 1.6	377	0.1050	1.82	8.6208	1.85
≥ 1.6	748	0.0884	1.58	7.7395	2.16

Figure 2.16 and Figure 2.17 combine a box plot and a Jitter graph to illustrate the dispersion of the observations of the ratio of the expected number of relocations of the product lot per day and the ratio of the expected number of relocated unit loads of the product lot per day. The "*" point indicates the average value of the categories defined by the range of RASL; the solid line passes through these average points.

As shown in the fourth column of Table 2.18, Figure 2.16 indicates the ratio of the expected number of relocation of the product lot per day decreases as RASL increases. Like the sixth column of Table 2.18, Figure 2.17 shows the percentage of expected number of relocated unit loads of the product lot per day increases as RASL increases.

To summarize Table 2.18, Figure 2.16, and Figure 2.17, the product lot with the lower RASL is relocated more frequently with smaller number of unit loads whereas the product lot with the higher RASL is relocated less frequently with larger number of unit loads. Notice, the product lot's operating cost is reduced by relocation when the saving in the floor's space cost by reduced honeycomb loss is higher than the increase in the material handing cost due to relocation. The lower inventory level guarantees the lower relocation cost and, consequently, provides a greater chance the savings is greater than the relocation cost, compared to a higher inventory level.

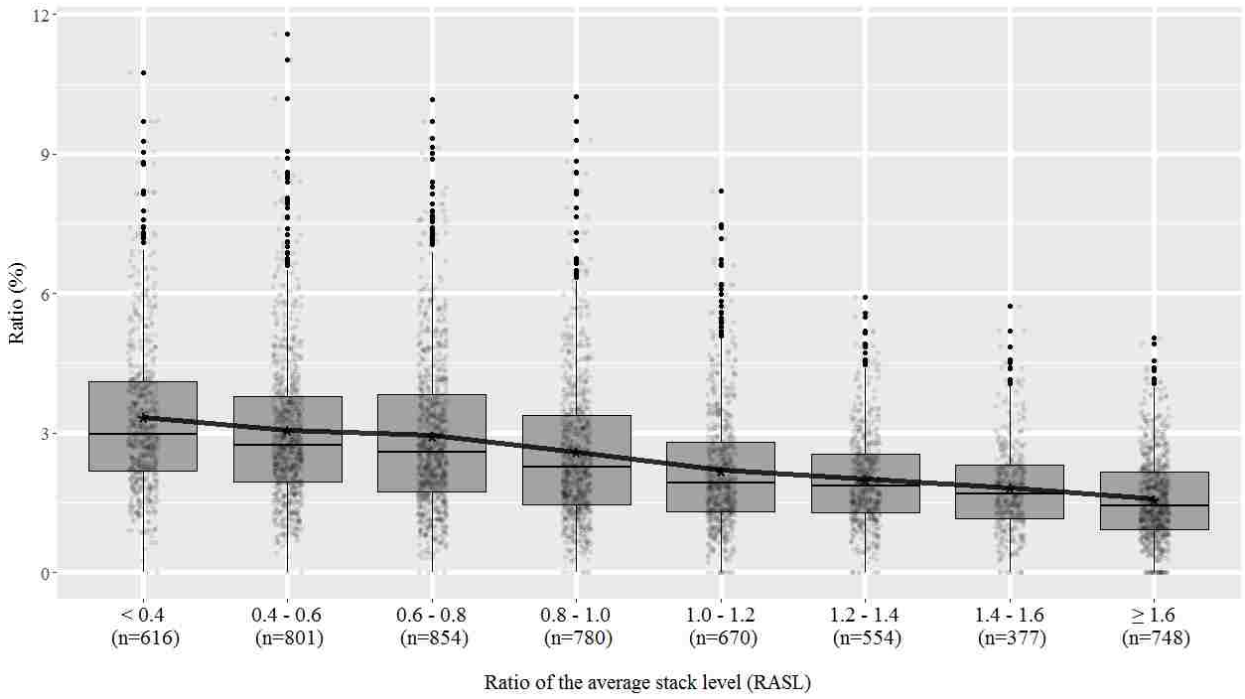

Figure 2.16: The ratio of the expected number of relocations of the product lot per day to the expected number of relocations per day as RASL changes

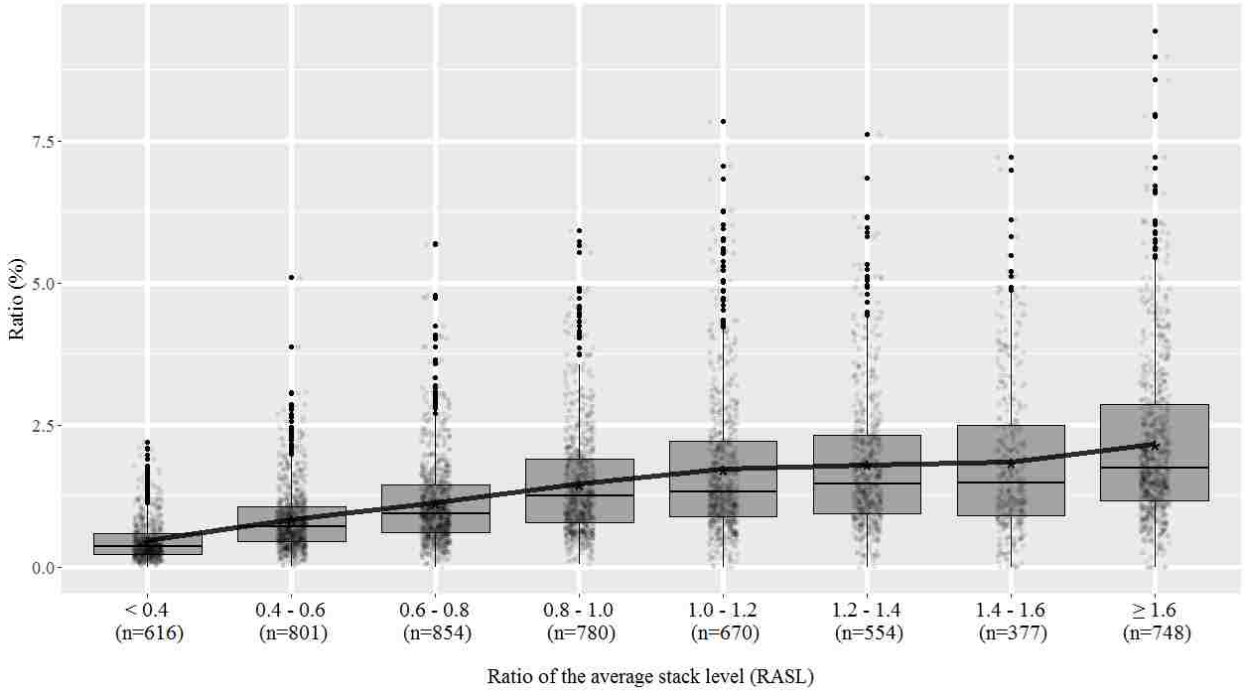

Figure 2.17: The ratio of the expected number of relocated unit loads of the product lot per day to the expected number of relocated unit loads per day as RASL changes

A possible hypothesis concerning the changing row depth is the designated row depth for the product lot only changes to a shallower row depth. It is not true. For example, consider the product lot of 6 stacks. If it is stored in the 5-deep storage area, the product lot incurs the honeycomb loss of 4 stack positions. If 6-deep and 4-deep storage areas are available, it is better to relocate it into a 6-deep storage area and save the floor space cost by reducing honeycomb loss. Notice 6-deep and 4-deep storage areas incur honeycomb loss of 0 stack position and 2 stack positions, respectively. In the experiment, 43,863 relocations are observed. Among them, 30,761 relocations (70.13%) occur from a deeper storage area to a shallower storage area and 13,102 relocations (29.87%) occur from a shallower storage area to a deeper storage area.

Another hypothesis is relocation only occurs when operating cost is reduced. It is true in the case of block stacking a single product lot. However, it is not true in the case of block stacking multiple product lots. Sometimes, a product lot is relocated without reducing floor space cost. In the experiment, 2,904 relocations (6.62%) increased honeycomb loss by 3.34 stack positions. The objective of relocation is to yield storage space to the product lot with a higher priority in order to achieve global optimization.

6. Conclusions

The first contribution of this paper is IP-BSMPwRuDD, the first optimization model for block stacking multiple products with changeable row depth under deterministic demand. Solving the model results in a DBS plan determining an assignment of product lots to storage areas each day given known inventory levels and daily demands over a planning horizon to minimize total operating cost. A DBS plan is distinguished from a conventional SBS plan determining an assignment of product lots to storage areas permanent over a time horizon. Compared to SBS, DBS requires less storage capacity and incurs less operating cost. DBS uses floor space

efficiently by timely relocation of product lots. It not only alleviates honeycomb loss and enables the product lot to yield occupied storage locations to another product lot if required. The savings in space cost by changing row depth is greater than the increase in material handling cost due to relocation; consequently, DBS decreases operating cost. The merit of DBS is magnified when storage capacity is relatively insufficient based on the inventory level.

The second contribution this paper is DH, the solution method based on a strategy of decomposing the original problem into smaller and easier-to-solve sub-problems. It consists of an upper-bounding procedure based on THD heuristics and a lower-bounding procedure based on LD heuristics. For practical-sized instances, DH solves corresponding optimization problems in a reasonable time and guarantees a feasible DBS plan. The small average and narrow range of the optimality gaps vouches for the quality of the solution. Notice, compared to DH, in solving practical-sized instances, the CPLEX branch-and-cut algorithm requires longer computation time and rarely provides a feasible solution.

The third contribution of this paper is the quantitative analysis of relocation behavior in a block stacking system. When the system has relatively lower storage capacity and higher average space utilization, relocation occurs more frequently. Relatively higher floor space unit cost and/or lower material handling unit cost support product lots being actively relocated. We can expect more unit loads to be relocated the higher space utilization. A product lot with a lower average inventory level is more likely to be relocated more frequently than a product lot with a higher average inventory level.

For future research, SDBS can be considered, first. It has a potential to be an alternative against DBS but requires more study. BSMPwRuDD with lot splitting is a very interesting but very challenging research problem. The solution of the problem would provide a more practical

DBS plan under a deterministic demand setting. Aggregating the problems of DBS optimization and replenishment scheduling and integrating the problems of DBS optimization and facility design are other interesting issues for future research.

7. Reference

Ahuja, R. K., Magnanti, T. L., and Orlin J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Upper Saddle River, NJ

Ashayeri, J., and Gelders, L. F. (1985). Warehouse design optimization. *European Journal of Operational Research*, 21(3), 285-294.

Bartholdi, J. J., and Hackman S. T. (2014). *Warehouse & Distribution Science*: release 0.96. Georgia Institute of Technology, Atlanta, GA. Retrieved from http://www.warehouse-science.com.

Berry, J. R. (1968). Elements of warehouse layout. *International Journal of Production Research*, 7(2), 105-121.

Boctor, F. F. (2010). Offsetting inventory replenishment cycles to minimize storage space. *European Journal of Operations Research*, 203, 321-325.

Bertsekas, D. P. (2003). *Nonlinear Programming*, 2nd edition. Athena Scientific, Belmont, MA.

Carlo, H. J., Vis, I. F. A., and Roodbergen, K. J. (2014). Storage yard operations in container terminals: Literature overview, trends, and research directions. *European Journal of Operational Research*, 235(2), 412-430.

De Koster, R. (2010). *Warehouse Math*. L. Kroon, T. L, R. Zuidwijk (eds.), Liber amicorum. In memoriam Jo van Nunen, 179-186, Dinalog Breda.

Derhami, S., Smith, J. S., and Gue, K. R. (2016). A simulation model to evaluate the layout for block stacking warehouse. *Progress in Material Handling Research:2016*, 14th International Material Handling Research Colloquium, http://www.mhi.org/cicmhe/colloquium.

Derhami, S., Smith, J. S., and Gue, K. R. (2017) Optimizing space utilization in block stacking warehouse. *International Journal of Production and Research*, 55(21), 6436-6452.

Fisher, M. L. (1985). An application oriented guide to Lagrangian relaxation. *Interfaces*, 15(2), 10-21

Fisher, M. L. (2004). The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 50(12), 1861-1871

Frangioni, A. (2005). About Lagrangian method in integer optimization. *Annals of Operations Research*, 139(1), 2005, 163-193.

Gavin, T. R. (1979). Keep your warehouse competitive. *Handling & Shipping Management*, 20, 34-40.

Geoffrion, A. M. (1974). Lagrangean relaxation for integer programming. *Mathematical Programming Study*, 2, 82-114.

Goetschalckx, M., and Ratliff, H. D. (1991). Optimal land depth for single and multiple products in block stacking storage systems. *IIE Transactions*, 23(3), 245-258.

Gu, J., Goetschalckx, M., and McGinnis L. F. (2010). Research on warehouse design and performance evaluation: A comprehensive review. *European Journal of Operations Research*, 203(3), 539-549

Guignard, M. and Kim, S. (1987). Lagrangean decomposition: A model yielding stronger Lagrangean bounds. *Mathematical Programming*, 39(2), 215-228.

Guinard, M. and Rosenwein, M. B. (1990). An application of Lagrangean decomposition to the resource-constrained minimum weighted arborescence problem. *Networks*, 20(3), 345-359.

Guta, B. (2003). Subgradient optimization methods in integer programming with an application to a radiation therapy problem. Doctoral dissertation. Kaiserslautern Technische Universität, Kaiserslautern, Germany

Hemmi, J. B. (1963). *An Investigation of the Influence of Warehouse Layout on Storage Costs*. Master's thesis. Georgia Institute of Technology, Atlanta, GA.

Kay, M. G. (2015). *Warehousing*. North Carolina State University, Raleigh, NC. Retrieved from http://www4.ncsu.edu/~kay/Warehousing.pdf.

Kind, D. A. (1965) Measuring warehouse space utilization. *Transportation and Distribution Management*, 5(7), 23-33.

Kind, D. A. (1975). Elements of space management. *Transportation and Distribution Management*, 15(2), 29-34.

Kooy, E. D. (1981). Making better use of available warehouse space. *Industrial Engineering*, 13(10), 26-30.

Larson, T. N., March, H., and Kusiak, A. (1997) A heuristic approach to warehouse layout with class-based storage. *IIE Transactions*. 29(4), 337-348.

Lee. H., Zhang, S., and White, J. A. (2016). The dynamic block stacking problem with random demand. *Proceedings of the 2016Industrial and System Engineering Research Conference*, *H. Yang, Z. Kong, and MD Sarder* (eds), Anaheim, CA.

Mao, K. Pan, Q., Chai, T., and Luh, P. B. (2015) An effective subgradient method for scheduling a steelmaking-continuous casting process. IEEE Transactions on Automation Science and Engineering, 12(3), 1140-1152.

Marsh, W. H. (1979). Elements of block storage design. *International Journal of Production Research*, 17(4), 377-394.

Martello, S. and Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc. New York, NY

Matson, J. O. (1982). *The Analysis of Selected Unit Load Storage Systems*. Doctoral dissertation. Georgia Institute of Technology, Atlanta, GA.

Matson, J. O., Sonnentag, J. J., White, J. A., and Imhoff, R. C. (2014). An analysis of block stacking with lot splitting. In *Proceedings of the 2014 Industrial and System Engineering Research Conference, Y. Guan and H. Liao* (eds), Montreal, Quebec, Canada, 479-506

Moder, J. J., and Thornton, H. M. (1965). Quantitative analysis of the factors affecting floor space utilization of palletized storage. *Journal of Industrial Engineering*, 16(1), 8-18.

Moon, I.K., Cha, B.C., and Kim, S.K. (2008). Offsetting inventory cycles using mixed integer programming and genetic algorithm. *International Journal of Industrial Engineering*, 15(3), 245-256.

Ouorou, A., Mahey, P., and Vial J.-Ph. (2000). A survey of algorithm for convex multicommodity flow problems. Management Science, 46(1), 126-147.

Peinhardt, M. A. F. (2003). *Integer multicommodity flows in Optical Networks*. Diplomarbeit, Technische Universität Berlin.

Rickles, H. V. and Elliott, K. A. (1985). Spreadsheet programming enables quick custom analyses of material handling problems. Industrial Engineering, 17(2), 80-85.

Roberts, S. D. (1968). *Warehouse Size and Design*. Doctoral dissertation. Purdue University, West Lafayette, IN.

Ross, P (1993). Basics of pallet storage systems: Part 1. *Material Handling Engineering*, 48(4), 68-69.

Sonnentag, J. J., Imhoff, R. C., White, J. A., and Matson J. O. (2014) A consideration of the block stacking multi-product storage problem. In *Proceedings of the 2014 Industrial and System Engineering Research Conference, Y. Guan and H. Liao* (eds), Montreal, Quebec, Canada, 3221-3230

Thornton, H. M. (1961). Factors Affecting Space Efficiency of Palletized Storage. Master's thesis. Georgia Institute of Technology, Atlanta, GA.

Tompkins, J. A., White, J. A., Bozer, Y. A., and Tanchoco, J. M. A. (2010) *Facility Planning,* 4th edition. John Wiley & Sons, Inc., New York, NY.

White, J. A., Sonnentag, J. J., and Matson J. O. (2013). New insights regarding block stacking. In *Proceedings of the 2013 Industrial and System Engineering Research Conference, A Krishnamurthy and W.K.V. Chan* (eds), San Juan, Puerto Rico, 1225-1234.

8. Appendices

8.1. Examples of the procedure updating $\overline{\mathbf{X}}$ in the THD heuristic

In this section, we assume the instance where T is five and use the following notations:

k	stage of the procedure updating $\overline{\mathbf{X}}$	$\widetilde{\mathbf{X}}_k(\bar{t})$	updated $\overline{\mathbf{X}}(\bar{t})$ by the solution of $\mathrm{THD}^3(\overline{\mathbf{X}}, \bar{t})$ solved at state k
$\overline{\mathbf{A}}$	lot-to-storage-area assignment defined by $\overline{\mathbf{X}}$	$\overline{\mathbf{A}}_k(\bar{t})$	updated $\overline{\mathbf{A}}(\bar{t})$ by $\widetilde{\mathbf{X}}_k(\bar{t})$
F, B	forward search, backward search		

Example 1 where backward search starts at day T

The following table show how $\overline{\mathbf{X}}$ and $\overline{\mathbf{A}}$ are updated and $\phi(t)$ changes by THD algorithm.

k	$\hat{t}$	$\overline{\mathbf{X}}$	$\overline{\mathbf{A}}$	$\phi(t)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, \bar{t})$	Improve	Search	THD algorithm	Related $\overline{\mathbf{A}}$
1	1	$\{\overline{\mathbf{X}}(1), \overline{\mathbf{X}}(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\overline{\mathbf{A}}(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{0,0,0,0,0}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 1)$	o	F	Line (14)	$\overline{\mathbf{A}}(5), \overline{\mathbf{A}}(2)$
2	2	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_1(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{0,1,0,0,1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 2)$	o	F	Line (14)	$\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(3)$
3	3	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_2(2), \widetilde{\mathbf{X}}_2(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \widetilde{\mathbf{A}}_2(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{1,1,1,0,1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 3)$	o	F	Line (14)	$\widetilde{\mathbf{A}}_2(2), \overline{\mathbf{A}}(4)$
4	4	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_2(2), \widetilde{\mathbf{X}}_3(3), \widetilde{\mathbf{X}}_3(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_3(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{1,1,1,1,1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 4)$	o	F	Line (14)	$\widetilde{\mathbf{A}}_3(3), \overline{\mathbf{A}}(5)$
5	5	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_2(2), \widetilde{\mathbf{X}}_2(3), \widetilde{\mathbf{X}}_4(4), \widetilde{\mathbf{X}}_4(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_3(3), \widetilde{\mathbf{A}}_4(4), \overline{\mathbf{A}}(5)\}$	{1,1,1,1,1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 5)$	x	F	Line (21)	$\widetilde{\mathbf{A}}_4(4), \widetilde{\mathbf{A}}_1(1)$
6	3	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_2(2), \widetilde{\mathbf{X}}_2(3), \widetilde{\mathbf{X}}_4(4), \widetilde{\mathbf{X}}_4(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_3(3), \widetilde{\mathbf{A}}_4(4), \overline{\mathbf{A}}(5)\}$	{1,1,1,-1,-1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 3)$	x	B	Line (26)	$\widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_4(4)$
7	2	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_2(2), \widetilde{\mathbf{X}}_2(3), \widetilde{\mathbf{X}}_4(4), \widetilde{\mathbf{X}}_4(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_3(3), \widetilde{\mathbf{A}}_4(4), \overline{\mathbf{A}}(5)\}$	{1,-1,-1,-1,-1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 2)$	x	B	Line (26)	$\widetilde{\mathbf{A}}_1(1), \widetilde{\mathbf{A}}_3(3)$
8	1	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_2(2), \widetilde{\mathbf{X}}_2(3), \widetilde{\mathbf{X}}_4(4), \widetilde{\mathbf{X}}_4(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_3(3), \widetilde{\mathbf{A}}_4(4), \overline{\mathbf{A}}(5)\}$	{1,-1,-1,-1,-1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 1)$	x	B	Line (26)	$\overline{\mathbf{A}}(5), \widetilde{\mathbf{A}}_2(2)$
9				{-1,-1,-1,-1,-1}				Line (9)	

Given $\overline{\mathbf{X}}$ and $\overline{\mathbf{A}}$ at the termination, each $\mathrm{THD}^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$ is visited as follows:

$\mathrm{THD}^3(\overline{\mathbf{X}}, \bar{t})$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 1)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 2)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 3)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 4)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 5)$
Related $\overline{\mathbf{A}}$	$\overline{\mathbf{A}}(5), \widetilde{\mathbf{A}}_2(2)$	$\widetilde{\mathbf{A}}_1(1), \widetilde{\mathbf{A}}_3(3)$	$\widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_4(4)$	$\widetilde{\mathbf{A}}_3(3), \overline{\mathbf{A}}(5)$	$\widetilde{\mathbf{A}}_4(4), \widetilde{\mathbf{A}}_1(1)$
Visited	at state 8	at state 7	at state 6	at state 4	at state 5

Example 2 where backward search starts after passing day T

The following table show how $\overline{\mathbf{X}}$ and $\overline{\mathbf{A}}$ are updated and $\phi(t)$ changes by THD algorithm.

k	$\hat{t}$	$\overline{\mathbf{X}}$	$\overline{\mathbf{A}}$	$\phi(t)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, \bar{t})$	Improve	Search	THD algorithm	Related $\overline{\mathbf{A}}$
1	1	$\{\overline{\mathbf{X}}(1), \overline{\mathbf{X}}(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\overline{\mathbf{A}}(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{0,0,0,0,0}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 1)$	o	F	Line (14)	$\overline{\mathbf{A}}(5), \overline{\mathbf{A}}(2)$
2	2	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_1(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{0,1,0,0,1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 2)$	o	F	Line (14)	$\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(3)$
3	3	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_2(2), \widetilde{\mathbf{X}}_2(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \widetilde{\mathbf{A}}_2(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{1,1,1,0,1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 3)$	o	F	Line (14)	$\widetilde{\mathbf{A}}_2(2), \overline{\mathbf{A}}(4)$
4	4	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_2(2), \widetilde{\mathbf{X}}_3(3), \widetilde{\mathbf{X}}_3(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_3(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{1,1,1,1,1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 4)$	o	F	Line (14)	$\widetilde{\mathbf{A}}_3(3), \overline{\mathbf{A}}(5)$
5	5	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_2(2), \widetilde{\mathbf{X}}_2(3), \widetilde{\mathbf{X}}_4(4), \widetilde{\mathbf{X}}_4(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_3(3), \widetilde{\mathbf{A}}_4(4), \overline{\mathbf{A}}(5)\}$	{1,1,1,1,1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 5)$	o	F	Line (14)	$\widetilde{\mathbf{A}}_4(4), \widetilde{\mathbf{A}}_1(1)$
6	1	$\{\widetilde{\mathbf{X}}_5(1), \widetilde{\mathbf{X}}_2(2), \widetilde{\mathbf{X}}_2(3), \widetilde{\mathbf{X}}_4(4), \widetilde{\mathbf{X}}_5(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_3(3), \widetilde{\mathbf{A}}_4(4), \widetilde{\mathbf{A}}_5(5)\}$	{1,1,1,1,1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 1)$	o	F	Line (14)	$\widetilde{\mathbf{A}}_5(5), \widetilde{\mathbf{A}}_2(2)$
7	2	$\{\widetilde{\mathbf{X}}_6(1), \widetilde{\mathbf{X}}_6(2), \widetilde{\mathbf{X}}_2(3), \widetilde{\mathbf{X}}_4(4), \widetilde{\mathbf{X}}_5(5)\}$	$\{\widetilde{\mathbf{A}}_6(1), \widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_3(3), \widetilde{\mathbf{A}}_4(4), \widetilde{\mathbf{A}}_5(5)\}$	{1,1,1,1,1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 2)$	x	F	Line (21)	$\widetilde{\mathbf{A}}_6(1), \widetilde{\mathbf{A}}_3(3)$
8	5	$\{\widetilde{\mathbf{X}}_6(1), \widetilde{\mathbf{X}}_6(2), \widetilde{\mathbf{X}}_2(3), \widetilde{\mathbf{X}}_4(4), \widetilde{\mathbf{X}}_5(5)\}$	$\{\widetilde{\mathbf{A}}_6(1), \widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_3(3), \widetilde{\mathbf{A}}_4(4), \widetilde{\mathbf{A}}_5(5)\}$	{-1,-1,1,1,1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 5)$	x	B	Line (26)	$\widetilde{\mathbf{A}}_4(4), \widetilde{\mathbf{A}}_6(1)$
9	4	$\{\widetilde{\mathbf{X}}_6(1), \widetilde{\mathbf{X}}_6(2), \widetilde{\mathbf{X}}_2(3), \widetilde{\mathbf{X}}_4(4), \widetilde{\mathbf{X}}_5(5)\}$	$\{\widetilde{\mathbf{A}}_6(1), \widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_3(3), \widetilde{\mathbf{A}}_4(4), \widetilde{\mathbf{A}}_5(5)\}$	{-1,-1,1,1,-1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 4)$	x	B	Line (26)	$\widetilde{\mathbf{A}}_3(3), \widetilde{\mathbf{A}}_5(5)$
10	3	$\{\widetilde{\mathbf{X}}_6(1), \widetilde{\mathbf{X}}_6(2), \widetilde{\mathbf{X}}_2(3), \widetilde{\mathbf{X}}_4(4), \widetilde{\mathbf{X}}_5(5)\}$	$\{\widetilde{\mathbf{A}}_6(1), \widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_3(3), \widetilde{\mathbf{A}}_4(4), \widetilde{\mathbf{A}}_5(5)\}$	{-1,-1,1,-1,-1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 3)$	x	B	Line (26)	$\widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_4(4)$
11				{-1,-1,-1,-1,-1}				Line (9)	

Given $\overline{\mathbf{X}}$ and $\overline{\mathbf{A}}$ at the termination, each $\mathrm{THD}^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$ is visited as follows:

$\mathrm{THD}^3(\overline{\mathbf{X}}, \bar{t})$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 1)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 2)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 3)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 4)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 5)$
Related $\overline{\mathbf{A}}$	$\widetilde{\mathbf{A}}_5(5), \widetilde{\mathbf{A}}_2(2)$	$\widetilde{\mathbf{A}}_6(1), \widetilde{\mathbf{A}}_3(3)$	$\widetilde{\mathbf{A}}_2(2), \widetilde{\mathbf{A}}_4(4)$	$\widetilde{\mathbf{A}}_3(3), \widetilde{\mathbf{A}}_5(5)$	$\widetilde{\mathbf{A}}_4(4), \widetilde{\mathbf{A}}_6(1)$
Visited	at stage 6	at stage 7	at stage 10	at stage 9	at stage 9

Example 3 where backward search starts before passing day T

The following table show how $\overline{\mathbf{X}}$ and $\overline{\mathbf{A}}$ are updated and $\phi(t)$ changes by THD algorithm.

k	$\hat{t}$	$\overline{\mathbf{X}}$	$\overline{\mathbf{A}}$	$\phi(t)$	$\text{THD}^3(\overline{\mathbf{X}}, \bar{t})$	Improve	Search	THD algorithm	Related $\overline{\mathbf{A}}$
1	1	$\{\overline{\mathbf{X}}(1), \overline{\mathbf{X}}(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\overline{\mathbf{A}}(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{0,0,0,0,0}	$\text{THD}^3(\overline{\mathbf{X}}, 1)$	o	F	Line (14)	$\overline{\mathbf{A}}(5), \overline{\mathbf{A}}(2)$
2	2	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_1(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{0,1,0,0,1}	$\text{THD}^3(\overline{\mathbf{X}}, 2)$	x	F	Line (21)	$\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(3)$
3	5	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_1(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{-1,-1,0,0,1}	$\text{THD}^3(\overline{\mathbf{X}}, 5)$	x	B	Line (26)	$\overline{\mathbf{A}}(4), \widetilde{\mathbf{A}}_1(1)$
4	4	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_1(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{-1,-1,0,0,-1}	$\text{THD}^3(\overline{\mathbf{X}}, 4)$	x	B	Line (26)	$\overline{\mathbf{A}}(3), \overline{\mathbf{A}}(5)$
5	3	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_1(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{-1,-1,0,-1,-1}	$\text{THD}^3(\overline{\mathbf{X}}, 3)$	x	B	Line (26)	$\overline{\mathbf{A}}(2), \overline{\mathbf{A}}(4)$
				{-1,-1,-1,-1,-1}				Line (9)	

Given $\overline{\mathbf{X}}$ and $\overline{\mathbf{A}}$ at the termination, each $\text{THD}^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$ is visited as follows:

$\text{THD}^3(\overline{\mathbf{X}}, \bar{t})$	$\text{THD}^3(\overline{\mathbf{X}}, 1)$	$\text{THD}^3(\overline{\mathbf{X}}, 2)$	$\text{THD}^3(\overline{\mathbf{X}}, 3)$	$\text{THD}^3(\overline{\mathbf{X}}, 4)$	$\text{THD}^3(\overline{\mathbf{X}}, 5)$
Related $\overline{\mathbf{A}}$	$\overline{\mathbf{A}}(5), \overline{\mathbf{A}}(2)$	$\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(3)$	$\overline{\mathbf{A}}(2), \overline{\mathbf{A}}(4)$	$\overline{\mathbf{A}}(3), \overline{\mathbf{A}}(5)$	$\overline{\mathbf{A}}(4), \widetilde{\mathbf{A}}_1(1)$
Visited	at stage 1	at stage 2	at stage 5	at stage 4	at stage 3

Example 4 where the solution of $\text{THD}^3(\overline{\boldsymbol{X}}, \bar{\boldsymbol{t}})$ at $\boldsymbol{\phi}(\bar{\boldsymbol{t}}) = 0$ cannot improve $\overline{\boldsymbol{X}}$

The following table show how $\overline{\mathbf{X}}$ and $\overline{\mathbf{A}}$ are updated and $\phi(t)$ changes by THD algorithm.

k	$\hat{t}$	$\overline{\mathbf{X}}$	$\overline{\mathbf{A}}$	$\phi(t)$	$\text{THD}^3(\overline{\mathbf{X}}, \bar{t})$	Improve	Search	THD algorithm	Related $\overline{\mathbf{A}}$
1	1	$\{\overline{\mathbf{X}}(1), \overline{\mathbf{X}}(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\overline{\mathbf{A}}(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{0,0,0,0,0}	$\text{THD}^3(\overline{\mathbf{X}}, 1)$	x	F	Line (23)	$\overline{\mathbf{A}}(5), \overline{\mathbf{A}}(2)$
2	2	$\{\overline{\mathbf{X}}(1), \overline{\mathbf{X}}(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\overline{\mathbf{A}}(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{-1,0,0,0,0}	$\text{THD}^3(\overline{\mathbf{X}}, 2)$	x	F	Line (23)	$\overline{\mathbf{A}}(1), \overline{\mathbf{A}}(3)$
3	3	$\{\overline{\mathbf{X}}(1), \overline{\mathbf{X}}(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\overline{\mathbf{A}}(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{-1,-1,0,0,0}	$\text{THD}^3(\overline{\mathbf{X}}, 3)$	x	F	Line (23)	$\overline{\mathbf{A}}(2), \overline{\mathbf{A}}(4)$
4	4	$\{\overline{\mathbf{X}}(1), \overline{\mathbf{X}}(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\overline{\mathbf{A}}(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{-1,-1,-1,0,0}	$\text{THD}^3(\overline{\mathbf{X}}, 4)$	o	F	Line (14)	$\overline{\mathbf{A}}(3), \overline{\mathbf{A}}(5)$
5	5	$\{\overline{\mathbf{X}}(1), \overline{\mathbf{X}}(2), \overline{\mathbf{X}}(3), \widetilde{\mathbf{X}}_4(4), \widetilde{\mathbf{X}}_4(5)\}$	$\{\overline{\mathbf{A}}(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \widetilde{\mathbf{A}}_4(4), \overline{\mathbf{A}}(5)\}$	{-1,-1,1,0,1}	$\text{THD}^3(\overline{\mathbf{X}}, 5)$	x	F	Line (21)	$\widetilde{\mathbf{A}}_4(4), \overline{\mathbf{A}}(1)$
6	3	$\{\overline{\mathbf{X}}(1), \overline{\mathbf{X}}(2), \overline{\mathbf{X}}(3), \widetilde{\mathbf{X}}_4(4), \widetilde{\mathbf{X}}_4(5)\}$	$\{\overline{\mathbf{A}}(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \widetilde{\mathbf{A}}_4(4), \overline{\mathbf{A}}(5)\}$	{-1,-1,1,-1,-1}	$\text{THD}^3(\overline{\mathbf{X}}, 3)$	x	B	Line (26)	$\overline{\mathbf{A}}(2), \widetilde{\mathbf{A}}_4(4)$
7				{-1,-1,-1,-1,-1}					

Given $\overline{\mathbf{X}}$ and $\overline{\mathbf{A}}$ at the termination, each $\mathrm{THD}^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$ is visited as follows:

$\mathrm{THD}^3(\overline{\mathbf{X}}, \bar{t})$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 1)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 2)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 3)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 4)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 5)$
Related $\overline{\mathbf{A}}$	$\overline{\mathbf{A}}(5), \overline{\mathbf{A}}(2)$	$\overline{\mathbf{A}}(1), \overline{\mathbf{A}}(3)$	$\overline{\mathbf{A}}(2), \widetilde{\mathbf{A}}_4(4)$	$\overline{\mathbf{A}}(3), \overline{\mathbf{A}}(5)$	$\widetilde{\mathbf{A}}_4(4), \overline{\mathbf{A}}(1)$
Visited	at stage 1	at stage 2	at stage 6	at stage 4	at stage 5

Example 5 where forward search restarts after backward search

The following table show how $\overline{\mathbf{X}}$ and $\overline{\mathbf{A}}$ are updated and $\phi(t)$ changes by THD algorithm.

k	$\hat{t}$	$\overline{\mathbf{X}}$	$\overline{\mathbf{A}}$	$\phi(t)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, \bar{t})$	Improve	Search	THD algorithm	Related $\overline{\mathbf{A}}$
1	1	$\{\overline{\mathbf{X}}(1), \overline{\mathbf{X}}(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\overline{\mathbf{A}}(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{0,0,0,0,0}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 1)$	o	F	Line (14)	$\overline{\mathbf{A}}(5), \overline{\mathbf{A}}(2)$
2	2	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_1(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{0,1,0,0,1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 2)$	x	F	Line (21)	$\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(3)$
3	5	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_1(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{-1,-1,0,0,1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 5)$	x	B	Line (26)	$\overline{\mathbf{A}}(4), \widetilde{\mathbf{A}}_1(1)$
4	4	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_1(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{-1,-1,0,0,-1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 4)$	x	B	Line (26)	$\overline{\mathbf{A}}(3), \overline{\mathbf{A}}(5)$
5	3	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_1(2), \overline{\mathbf{X}}(3), \overline{\mathbf{X}}(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(2), \overline{\mathbf{A}}(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{-1,-1,0,-1,-1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 3)$	o	B	Line (16)	$\overline{\mathbf{A}}(2), \overline{\mathbf{A}}(4)$
6	4	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_1(2), \widetilde{\mathbf{X}}_5(3), \widetilde{\mathbf{X}}_5(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(2), \widetilde{\mathbf{A}}_5(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{-1,1,0,1,-1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 4)$	x	F	Line (21)	$\widetilde{\mathbf{A}}_5(3), \overline{\mathbf{A}}(5)$
7	2	$\{\widetilde{\mathbf{X}}_1(1), \widetilde{\mathbf{X}}_1(2), \widetilde{\mathbf{X}}_5(3), \widetilde{\mathbf{X}}_5(4), \overline{\mathbf{X}}(5)\}$	$\{\widetilde{\mathbf{A}}_1(1), \overline{\mathbf{A}}(2), \widetilde{\mathbf{A}}_5(3), \overline{\mathbf{A}}(4), \overline{\mathbf{A}}(5)\}$	{-1,1,-1,-1,-1}	$\mathrm{THD}^3(\overline{\mathbf{X}}, 2)$	x	B	Line (26)	$\widetilde{\mathbf{A}}_1(1), \widetilde{\mathbf{A}}_5(3)$
8				{-1,-1,-1,-1,-1}					

Given $\overline{\mathbf{X}}$ and $\overline{\mathbf{A}}$ at the termination, each $\mathrm{THD}^3(\overline{\mathbf{X}}_{\hat{t}}, \bar{t})$ is visited as follows:

$\mathrm{THD}^3(\overline{\mathbf{X}}, \bar{t})$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 1)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 2)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 3)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 4)$	$\mathrm{THD}^3(\overline{\mathbf{X}}, 5)$
Related $\overline{\mathbf{A}}$	$\overline{\mathbf{A}}(5), \overline{\mathbf{A}}(2)$	$\widetilde{\mathbf{A}}_1(1), \widetilde{\mathbf{A}}_5(3)$	$\overline{\mathbf{A}}(2), \overline{\mathbf{A}}(4)$	$\widetilde{\mathbf{A}}_5(3), \overline{\mathbf{A}}(5)$	$\overline{\mathbf{A}}(4), \widetilde{\mathbf{A}}_1(1)$
Visited	at stage 1	at stage 7	at stage 5	at stage 6	at stage 3

8.2. Procedure of generating an instance randomly

In generating instances, details of cases are set randomly once an instance type is specified by the number of product lots, the number of row depth types, and the time horizon.

At first, according to the number of row depth types, the set of row depth types are defined as shown in Table 2.19. We assume a storage area having the unique depth and, thus, the number of storage areas equals the number of row depth types. The layout of storage areas is such that shallower storage areas are located inside and deeper storage areas outside. We suppose all storage areas have the same number of row positions. In determining the number of row positions, we determine the minimum number of required row positions to accommodate unit loads of product lots over the time horizon without violating dynamic block stacking rules. Thus, the number of row positions of the layout of each instance is set as the objective function value of the following problem.

$$\min P$$

subject to

$$\sum_{l=1}^{L}\sum_{r=1}^{R} n_{r,t}^{l} x_{r,t}^{l} \leq P, \qquad r = 1, \dots, R \text{ and } t = 1, \dots, T$$

$$x_{r,t}^{l} \in \{0,1\}, \quad r = 1, \dots, R \text{ and } t = 1, \dots, T$$

where $x_{r,t}^{l}$=1 if lot l is stored in d_r-deep storage area for day t, 0 otherwise.

Table 2.19: Set of row depth types according the number of row depth types

Number of row depth types	Set of row depth types
4	{2, 3, 5, 10}
5	{2, 3, 5, 10, 15}
6	{2, 3, 5, 10, 15, 20}
7	{2, 3, 5, 10, 12, 15, 20}
8	{2, 3, 5, 8, 10, 12, 15, 20}

Next, we randomly set each product lot's inventory cycle length. It is selected equally likely from a designated set of values. Table 2.20 shows which set of inventory cycle lengths is considered according to the time horizon. For instance, when the time horizon is 20, a lot's inventory cycle length is equally likely to be 5, 10, or 20.

Table 2.20: Set of inventory cycle length according to the time horizon

Time horizon	Inventory cycle Length
20	{5, 10, 20}
30	{5, 6, 10, 15, 30}
40	{5, 8, 10, 20, 40}
60	{5, 6, 10, 12, 15, 20, 30, 60}
90	{5, 6, 9, 10, 15, 18, 30, 45, 90}
180	{5, 6, 9, 10, 12, 15, 18, 20, 30, 36, 45, 60, 90, 180}

Once the inventory cycle length is randomly set for product lots, we randomly select stack height and daily demand for each product lot from a given set of equally likely values. For the stack height, we consider the set of $\{2,3,4\}$, regardless of other features of the product lot. The set of daily demands is established, based on cycle length, daily inventory level, and reasonable capacity of storage areas; values range from 1 to 200.

Figure 2.18 shows how a product lot's daily inventory level can be computed over a time horizon under deterministic demand and a fixed order quantity. Notice the two tables at the left in Figure 2.18. The upper table shows $Q^1 = 12$, $D^1 = 4$, and $T^1 = 3$ and the lower table shows $Q^2 = 30$, $D^2 = 5$, and $T^2 = 6$. Thus, the inventory level of product lots 1 and 2 repeat on 3-day cycle and 6-day cycle, respectively. Based on $T^1 = 3$ and $T^2 = 6$, $T = 6$ and $\sigma^1 = 2$ and $\sigma^2 = 1$. The three tables at the right in Figure 2.18 show how the collection $\{I_t^1, I_t^2\}$ repeats on a 6-day cycle. The first, second, and third tables are developed based on $\{I_1^1 = 12, I_1^2 = 30\}$, $\{I_1^1 = 4, I_1^2 = 30\}$, and $\{I_1^1 = 8, I_1^2 = 30\}$.

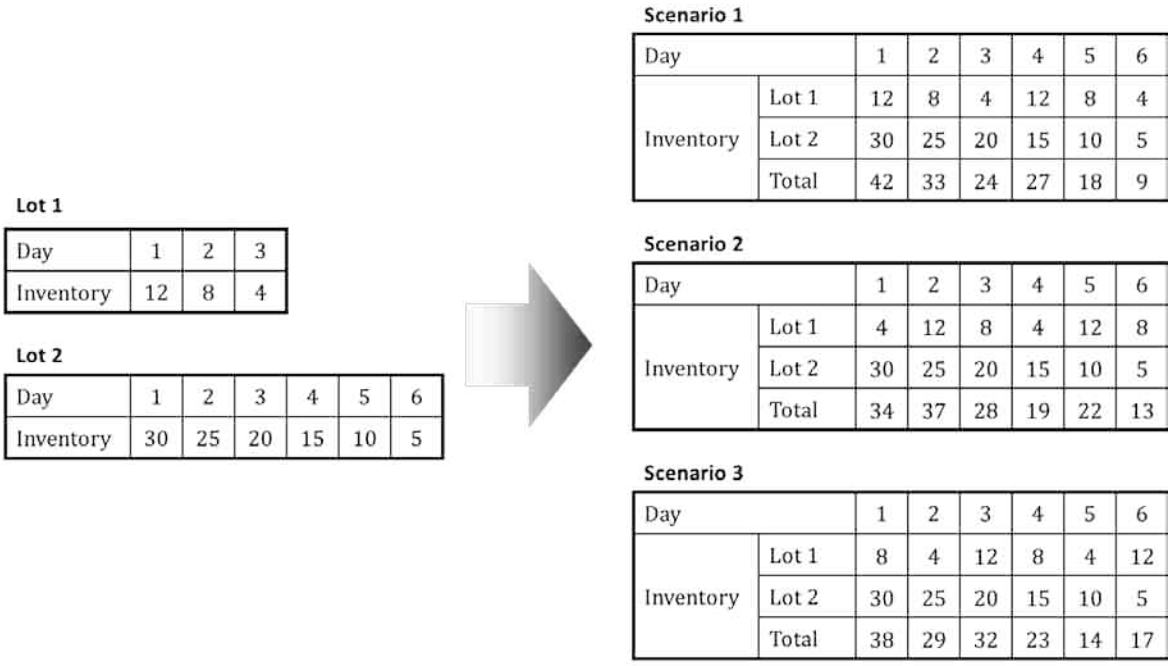

Figure 2.18: Inventory flow of multiple lots

In this research, we assume $\{I_1^l: l = 1, \dots, L\}$ is set to minimize the maximum aggregate inventory level of the product lots. In the example of Figure 2.18, the maximum aggregate inventory level in each table is 42, 37, and 38, respectively. Therefore, $\{I_1^1 = 4, I_1^2 = 30\}$ is considered in planning dynamic block stacking. For more details about the significance of the replenishment schedule in planning a block stacking storage system for multiple lots, see Matson (1982). To determine $\{I_1^l: l = 1, \dots, L\}$ of an instance, we solve the optimization problem of offsetting inventory cycles. For more details of the problem, refer to Moon *et al*. (2008) and Boctor (2010).

Table 2.21 shows values of parameters concerning unit load dimensions, vehicle velocities, working time to pick up and put down a unit load. Most are from White *et al*. (2013).

Table 2.21: Parameters of the instances

Parameter	Value	Parameter	Value	Parameter	Value
L	4 ft	A	13 ft	v_{ha}	240 fpm
W	3.5 ft	H	4.5 ft	v_{hr}	80 fpm
c	0.75 ft	M	0.5 min	v_v	50 fpm

In the numerical experiment, we consider different combinations of cost parameters. From an expert, in January 2019, the floor space unit cost and the material handling unit cost are given as follows:

Floor space unit cost: \$50/ft^2 /year ~ \$60/ft^2/year

Material handling unit cost: \$24/hr ~ \$29/hr

Material handling unit cost includes labor cost and material handling equipment cost. Floor space unit cost is converted to the cost per day by dividing annual lease cost per square foot by 260 working days and material handling unit cost is converted to the cost per minute by dividing the hourly cost of labor and rental cost of material handling equipment by 60 minutes per hour as follows:

Floor space cost: \$0.1923/ft^2 /day ~ \$0.2307/ft^2/day

Material handling cost: \$0.4/min ~ \$0.4833/min

Based on the ranges of costs, we considered five combinations of costs given in Table 2.22. . The combinations of the medium floor space cost and the medium material handling cost, (\$0.22/ft^2 /day, \$0.44/min), are used in all experiments excluding experiments comparing the results under different cost parameters.

Table 2.22: Combination of costs parameters

Feature		FSC	MHC
Low floor space cost	Low material handling cost	\$0.20/ft^2 /day	\$0.40/min
Low floor space cost	High material handling cost	\$0.20/ft^2 /day	\$0.48/min
Medium floor space cost	Medium material handling cost	\$0.22/ft^2 /day	\$0.44/min
High floor space cost	Low material handling cost	\$0.24/ft^2 /day	\$0.40/min
High floor space cost	High material handling cost	\$0.24/ft^2 /day	\$0.48/min

CHAPTER 3. Block Stacking Multiple Products with Relocation under Stochastic Demand

Abstract

This research investigates the problem of optimizing block stacking multiple products with relocation under stochastic demand to minimize total operating cost over a time horizon. Assuming changeable row depth instead of permanent row depth, this paper is distinguished from conventional block stacking studies. The problem is formulated as a discrete time finite horizon Markov decision process model, supposing the recursive daily situation of determining the assignment of product lots to storage areas for a day based on uncertain daily demand and observed system information. To tackle computational intractability in solving practical-sized instances, we develop a heuristic solution approach taking an on-line manner by instantly determining an action for a single observed state rather than an off-line manner by predetermining an action for every state. In the heuristic, we apply a simulation-based sampling technique to avoid searching all reachable future states from the observed state. Additionally, the action determination problem is formulated as a generalized assignment problem and solved by a branch-and-cut algorithm, enabling avoiding enumeration of all possible actions. The results of numerical experiments verify the reliability and efficiency of the heuristic in solving practical-sized instances.

Keyword

Block stacking, stochastic demand, changeable row depth, Markov decision process, on-line manner, sampling

1. Introduction

A block stacking system is a commonly used storage method in which unit loads are stacked on top of each other and the stacks are aligned in storage rows. A block stacking system consists of storage areas having the same or different depths and lengths; stack positions and row positions are explicitly designated in each storage area. Figure 3.1 illustrates a block stacking storage system consisting of four storage areas of the same length but different depths. The depth and length of a storage area specify the number of stack positions in a single row and the number of rows, respectively. Because the configuration of storage areas can be specified simply by painting lines on the floor, the design and layout of a block stacking storage system can be modified relatively easily (Ross, 1993).

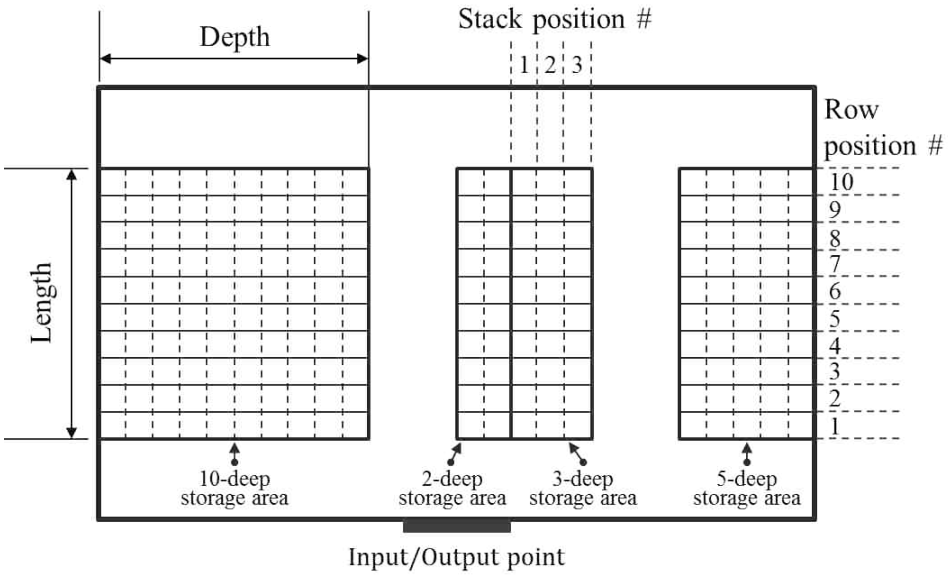

Figure 3.1: An example of the block stacking system

Block stacking uses limited or no supporting equipment; consequently, a block stacking system benefits from low investment cost but suffers from inconvenience in handling unit loads. Generally, block stacking is effective when a warehouse handles a small number of large-volume

product lots. Thus, it is usually adopted in facilities storing appliances, food and beverages, household products, tires, bags of potting mix and fertilizer, construction materials, and so on (Matson and White, 1982, Sonnentag et al, 2014).

In order to avoid possible blockage among unit loads during their retrieval, it is not allowed to consolidate unit loads of different products or different lots of the same product. Thus, once a row position is occupied by a product lot, openings in a partially filled row position cannot be used until the entire row becomes vacated. Space losses due to underutilized row positions are referred to as honeycombing. For more details of block stacking, refer to Bartholdi and Hackman (2014) and Tompkins *et al*. (2010).

In operating a block stacking system, it is important to increase space utilization by reducing honeycomb loss. The extent of honeycombing incurred by a product lot is decided by the inventory level, the row depth (i.e., the maximum number of stacks in a row), and the stack height (i.e., the maximum number of unit loads in a stack). For example, consider a product lot having inventory of 9 unit loads and being stored in a 10-deep row. Assuming a stack height of 2 unit loads, honeycombing of 11 storage slots (11 = 2*10 – 9) or 5 stack positions (5 = 10 – $\lceil 9/2 \rceil$) is incurred. Among the three elements, the inventory level is not fully controllable and the stack height is predetermined considering the ceiling height, choice of storage/retrieval equipment, and load crushing. On the other hands, the row depth can be adjusted during a product lot's storage life by relocating its remaining unit loads. Consequently, given the inventory level and the stack height, we can reduce the honeycomb loss by positioning a product lot in a storage area having an appropriate depth.

Existing studies on block stacking developed mathematical models to determine the permanent row depth for the product lot with the objective of minimizing operating cost. Their

findings provided significant insight regarding the design of a block stacking system. However, by incorporating an assumption of permanent row depth, they did not address operational decision making issues and ignored the flexibility of block stacking.

In this research, we deal with dynamic block stocking (DBS) problem where a product lot can be relocated to change an assigned row depth. Changing row depth can reduce honeycomb loss and floor space cost, but relocating product lots incurs additional material handing cost. This motivated us to solve an optimization problem by determining an assignment of product lots to storage areas to minimize operating cost, including floor space cost and material handling cost.

Extending our previous work on the single product lot DBS problem (Lee *et al*. 2016), this research focuses on block stacking multiple products with relocation under stochastic demand (referred to as BSMPwRuSD). A solution of BSMPwRuSD, referred to as the DBS plan, determines which storage area should be chosen for each product lot to minimize the expected operating cost under a given configuration for the block stacking system. The BSMPwRuSD problem is framed as an infinite-horizon discrete-time Markov decision process (MDP) model. In order to solve practical-sized instances, we develop a solution procedure taking an on-line manner by instantly determining an action for a single observed state rather than an off-line manner by predetermining an action for every state.

Our main contributions are two-fold. First, we believe this is the first study to reflect the flexibility of block stacking in real-world operations by exploring block stacking of multiple products with changeable row depths under stochastic demand. Second, a solution method is proposed, which guarantees reliability and efficiency in solving practical-sized instances of the BSMPwRuSD problem.

The remainder of this chapter is organized as follows. In Section 2, the relevant block stacking literature is reviewed. Section 3 clarifies the assumption of BSMPwRuSD and presents the MDP model. In Section 4, we introduce a solution procedure based on the on-line manner and different ways of estimating future operating cost. Section 5 provides the results of numerical experiments and validates the reliability and efficiency of the solution procedure developed. We conclude in Section 6, summarizing research findings and suggesting research topics for future studies.

2. Literature Review

The applications of block stacking storage can be found in the field of warehouse design and control and in the management of container terminals (Accorsi *et al*. 2017). Because their operating principles are fundamentally different, research issues arising in the domain of block stacking in a warehouse and in a container terminal are distinct from each other. Interestingly, some papers (Yang and Kim, 2006; Kim and Hong, 2006; Petering and Hussein, 2013; and Jang *et al*. 2013) studied the problem of relocating containers in a block stacking system of the container terminal but they did not overlap with our research. Therefore, the review is limited to literature on block stacking in a warehouse. For those interested in research issues concerning block stacking in a container terminal, refer to Carlo *et al*. (2014). Hereafter, unless otherwise noted, block stacking indicates block stacking in a warehouse.

Within the warehouse research domain, previous block stacking studies can be categorized into either block stacking design or block stacking operations. Figure 3.2 illustrates this classification. Most previous papers are classified as the study of block stacking design. Comprehensive literature review on block stacking can be found in Ashayeri *et al* (1985) and Gu *et al*. (2010).

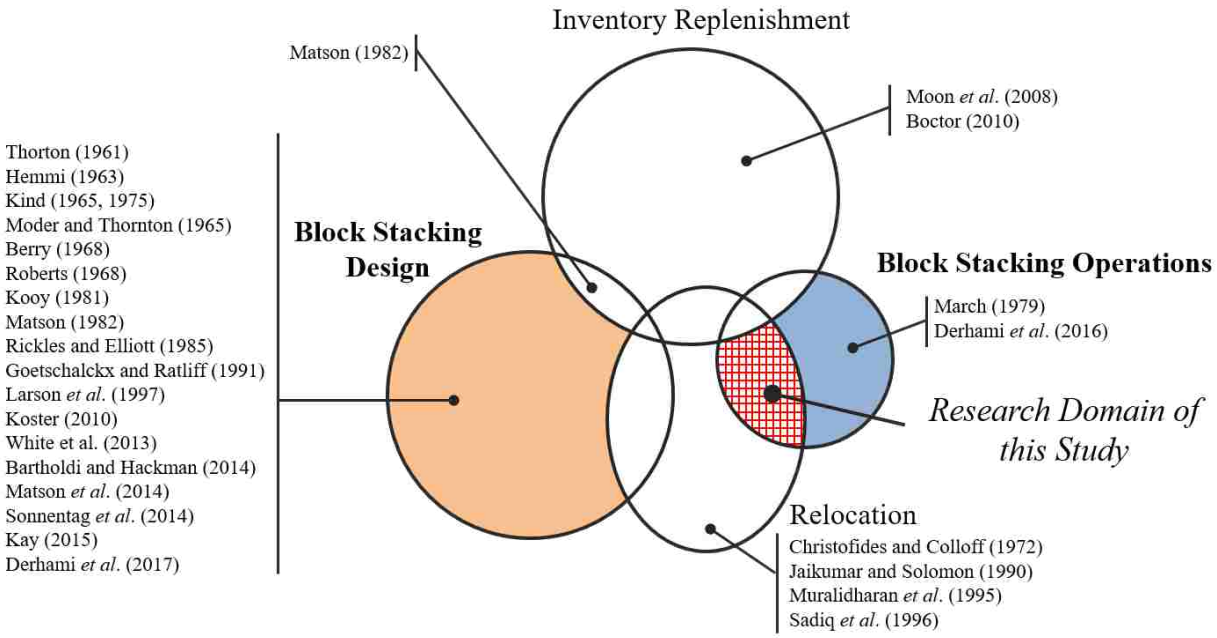

Figure 3.2: A graphical representation of research domains of warehouse literature related to block stacking multiple products with relocation under stochastic demand

Block stacking design research has attempted to determine a single or multiple row depth(s) for a single or multiple product lot(s) to achieve the objective of optimizing space utilization and reducing material handling. Their findings provided significant insight into designing a block stacking system. Table 3.1 categorizes published academic literature on block stacking design based on the output of each paper's optimization problem. The abbreviations in the "solution" column are defined as follows:

- SL single row depth for layout satisfying a given storage population
- SS single row depth for a single product lot
- SM single row depth for multiple product lots
- MS multiple row depths for a single product lot
- MM multiple row depths for multiple product lots
- CS changeable row depth for a single product lot
- CM changeable row depths for multiple product lots

Table 3.1: Categorization of published academic literature on block stacking design

Solution	Paper		
SL	Thorton (1961)	Hemmi (1963)	Moder and Thornton(1965)
	Berry (1968)	Rickles and Elliott (1985)	
SS	Kind (1965, 1975)	Matson (1982)	White et al. (2013)
SM	Matson (1982)	Koster (2010)	Bartholdi and Hackman (2014)
	Sonnentag *et al*. (2014)	Kay (2015)	Derhami *et al*. (2017)
MS	Kooy (1981)	Matson (1982)	Goetschalckx and Ratliff (1991)
	White et al. (2013)	Matson *et al*. (2014)	
MM	Roberts (1968)	Matson (1982)	Goetschalckx and Ratliff (1991)
	Larson *et al*. (1997)	Sonnentag *et al*. (2014)	
CS	Lee et al. (2016)		
CM	This paper		

Only a few papers investigated block stacking from an operational viewpoint. Marsh (1979) developed a simulation model to analyze the operational problem of block stacking under stochastic demand. Derhami, Smith, and Gue (2016) constructed a discrete-event simulation model to evaluate space utilization and travel cost of a given layout of a block stacking system. They assumed inbound events of unit loads following a non-stationary replenishment rate defined by stochastic production time and outbound events following a deterministic schedule.

All aforementioned studies of block stacking design and operations assumed the assigned row depth(s) for a product lot is permanent and did not consider changeable row depth by relocation. Many papers recognized the requirement of relocation in operating a block stacking system but we were unable to locate any published papers on the topic, excluding Lee *et al*. (2016). We found four papers addressing the issue of relocating unit loads in other storage systems (Christofides and Colloff, 1972; Jaikumar and Solomon, 1990; Muralidharan *et al.* 1995; Sadiq *et al*. 1996); they investigated problems where items of increasing-demand are relocated closer to the I/O point and showed the relocation's benefit of reduced retrieval travel time.

Inventory replenishment is a research area closely related to block stacking planning and operations. Matson (1982) noted the significance of the replenishment schedule in planning a block stacking storage system for multiple product lots. Moon *et al*. (2008) and Boctor (2010) provided efficient solution procedures for mathematical models to minimize the maximum inventory level over a time horizon.

To deal with BSMPwRuSD, we adopt MDP's systematic framework and scheme for solving sequential decision making problem with uncertain outcomes. Gong and De Koster (2011) conducted a comprehensive review of stochastic research in warehouse operations. They concluded no study has occurred adopting a MDP model with regard to warehouse operations. Among active research domains of MDP applications, inventory management is relatively close to this research. Assuming uncertain demand, a basic MDP model of inventory management determines whether and how much to order in a given inventory level state. Fianu and Davis (2018) and Ouaret, Kenne, and Gharbi (2018) recently studied inventory management with MDP.

To summarize, our research is unique, compared to previous studies of block stacking. Our study is distinct from conventional research on block stacking by introducing changeable row depth; by assuming stochastic demand, it is distinct from CHAPTER 2's research on BSMPwRuDD; and it is distinct from Lee *et al*. (2016) by considering multiple product lots. Additionally, to the best of our knowledge, this is the first work applying MDP in addressing storage system operations.

3. MDP Model of DBS under Stochastic Demand

In this study, DBS under stochastic demand is framed as a discrete-time and finite-horizon MDP model that represents the recursive daily decision of assigning product lots to storage areas based on uncertain daily demand and observed system information, product lots' inventory levels and

present storage areas. We refer to the model as MDP-DBS. We address the assumptions made in developing MDP-DBS in Section 3.1 and define key elements of MDP-DBS in Section 3.2. The notation of Table 3.2 is considered in this section.

Table 3.2: Notation for developing MDP-DBS

Notation	*Description*
L, l	number of lots considered, index of lot
B	number of storage areas considered including permanent and temporary storage area
b	index of storage area, $1 \leq b < B$ indicates a permanent storage area inside block stacking system and $b = B$ indicates the leased temporary storage area outside block stacking system
d_b	depth of storage area b, measured in the number of unit loads, such as $\begin{cases} 0 < d_b < \infty, & b = 1,2,\dots,B-1 \\ d_b = \infty, & b = B \end{cases}$.
N_b	capacity of storage area b, measured in the number of row positions, such as $\begin{cases} 0 < N_b < \infty, & b = 1,2,\dots,B-1 \\ N_b = \infty, & b = B \end{cases}$.
z^l	height of the stack of lot l, measured in the number of unit loads
Q^l	order quantity of lot l, measured in the number of unit loads
D^l	daily demand of product lot l, discrete random variable
$D^l_{\max}$	maximum daily demand of lot l

3.1. Assumption of MDP-DBS

A fundamental assumption of BSMPwRuSD is stochastic demand. We assume probability distributions of D^l, l=1,…,L are known and they are independent and identically distributed. Consequently, the joint probability mass function of D^l, l=1,…,L is defined as follows:

$$\Pr(D^1{=}k^1, \dots, D^L{=}k^L) = \Pr(D^1 = k^1) \dots \Pr(D^L = k^L) = \prod_{1 \leq l \leq L} \Pr(D^l = k^l) \qquad (3.1)$$

Because the exact inventory levels and daily demands are unknown in advance, in solving BSMPwRuSD, we find a daily DBS plan minimizing the total expected operating cost computed based on possible inventory levels of all product lots and their probabilities over a planning

horizon. In developing the MDP-DBS, we assume the following sequential decision making model. At the end of a business day, a manager monitors the situation of a block stacking system, including inventory levels and storage areas of all product lots. Based on the observed information, the manager determines a DBS plan for the following business day. If required, a product lot is immediately replenished and relocated within non-business hours during the present day, incurring an operating cost. The system then evolves into a new state at the end of the next business day when the manager, again, determines a DBS plan based on the state. This procedure is repeated daily.

We assume a block stacking system with finite storage capacity. If required storage space exceeds the capacity, a temporary storage area is leased. The temporary storage area has infinite storage capacity and is more expensive compared to the permanent storage area within the system.

3.2. Elements of MDP-DBS

In this section, we define five key elements: decision epoch, state, action, transition probability, and rewards of the MDP-DBS model (Puterman, 2005).

3.2.1. Decision epoch and period

Decision epochs are points of time where decisions are made. When modeling the discrete time problem, time is partitioned into regular intervals called *periods*; decision epochs exist between consecutive periods.

T indicates the set of decision epochs and t denotes an element of T. We assume an infinite time horizon; thus, $T = \{1,2,3, \dots\}$. Figure 3.3 illustrates decision epochs and periods of the MDP model of BSMPwRuSD.

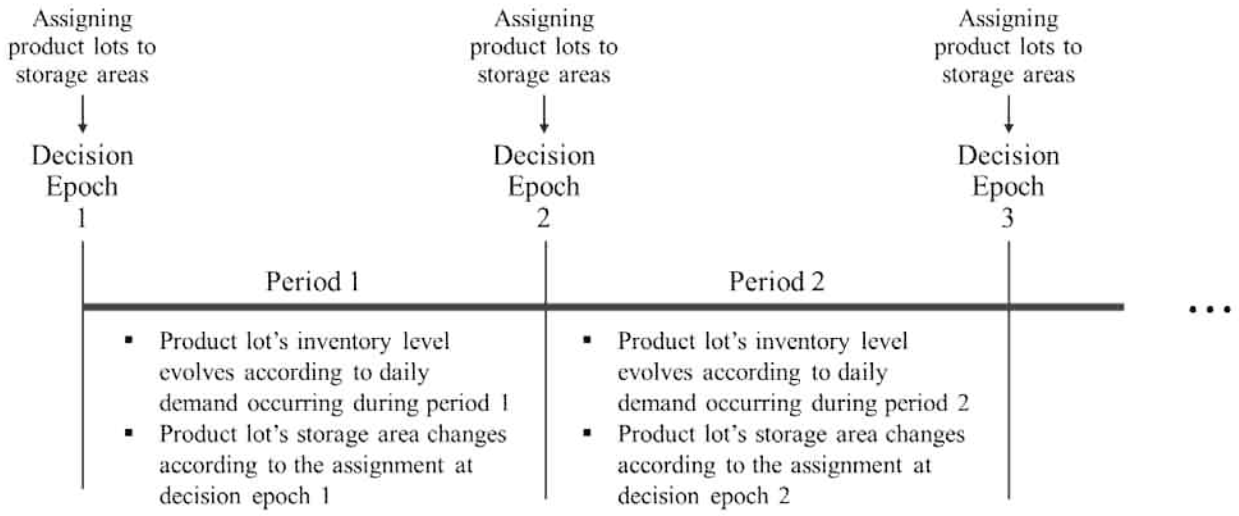

Figure 3.3: Decision epochs and periods of the MDP-DBS

3.2.2. Set of states

State is a concise description of the system at a decision epoch, providing information critical in decision making. In MDP-DBS, state represents inventory levels and storage areas of all product lots at the decision epoch. We use the notation of Table 3.3 to express hierarchical states of MDP-DBS.

Table 3.3: Notation of states of MDP-DBS

Notation	*Description*
i^l	inventory level of product lot l, $0 \leq i^l \leq Q^l$ (value of i^l indicates number of unit loads of product lot l at the decision epoch.)
r^l	storage area of product lot l, $1 \leq r^l \leq B$ (value of r^l indicates index of the storage area where unit loads of product lot l are stored at the decision epoch.)
s^l	aggregate state of product lot l, $s^l = (i^l, r^l)$
i	inventory state of the block stacking system, $\mathbf{i} = (i^1, i^2, \dots, i^L)$
r	storage area state of the block stacking system, $\mathbf{r} = (r^1, r^2, \dots, r^L)$
s	aggregate state of the block stacking system, $\mathbf{s} = (s^1, s^2, \dots, s^L)$

Let I^l be the set of possible inventory level states of product lot l and **I** be the set of possible inventory level states of the system. When i^l indicates the element of **i** corresponding to product lot l, **I** is defined as follows:

$$\mathbf{I} = \{\mathbf{i} | i^l \in I^l \text{ where } 1 \leq l \leq L\} \tag{3.2}$$

The cardinality of **I** is computed by

$$|\mathbf{I}| = \prod_{1 \leq l \leq L} |I^l| = \prod_{1 \leq l \leq L} (1 + Q^l) \tag{3.3}$$

Let $R_{i^l}^l$ represent the set of feasible storage areas of product lot l given i^l and $\mathbf{R_i}$ indicate the set of feasible storage area states of the system given **i**. When r^l indicates the element of **r** corresponding to product lot l, $\mathbf{R_i}$ is defined as follows:

$$\mathbf{R_i} = \left\{ \mathbf{r} \middle| r^l \in R_{i^l}^l \text{ where } 1 \leq l \leq L \text{ and } \sum_{\substack{1 \leq l \leq L \\ r^l = b}} \hat{n}_{i^l,b}^l \leq N_b \text{ for } 1 \leq b \leq B \right\} \tag{3.4}$$

where

$$\hat{n}_{i^l,b}^l = \left\lceil \frac{i^l}{d_b z^l} \right\rceil. \tag{3.5}$$

Note the value of $\hat{n}_{i^l,b}^l$ represents the required number of row positions to store i^l unit loads in storage area b (or d_b-deep storage area) and N_b is the capacity of storage area b measured in row positions. Thus, the inequality condition in (3.4) represents the capacity constraint of each storage area. Because the temporary storage area has infinite storage capacity ($N_B = \infty$), there exists at least one **r** satisfying the inequality condition in (3.4). Thus, the range of $\mathbf{R_i}$ is defined as follows:

$$1 \leq |\mathbf{R_i}| \leq B^L. \tag{3.6}$$

Let **S** be the set of possible aggregate states of the system defined as follows:

$$\mathbf{S} = \{\mathbf{s} | \mathbf{i} \in \mathbf{I} \text{ and } \mathbf{r} \in \mathbf{R_i}\}. \tag{3.7}$$

Note **s** can be expressed as the Cartesian product of **i** and **r** such as $\mathbf{s} = \mathbf{i} \times \mathbf{r}$. From $|\mathbf{I}|$ and $|\mathbf{R_i}|$, the range of $|\mathbf{S}|$ is defined as follows:

$$\prod_{1 \le l \le L} (1 + Q^l) \le |\mathbf{S}| \le B^L \prod_{1 \le l \le L} (1 + Q^l) \tag{3.8}$$

We assume **S** is stationary over a time horizon.

By introducing the concept of the feasible storage area state of the system in defining the set of the aggregate states of the system, we can reduce the number of elements in **S**. Let $\tilde{\mathbf{S}}$ be the set of the aggregate states of the system defined regardless of the feasibility of storage area state. The number of elements in $\tilde{\mathbf{S}}$ is $B^L \prod_{1 \le l \le L} (1 + Q^l)$ and equals the maximum value of $|\mathbf{S}|$. Thus, the inequality of $|\mathbf{S}| \le |\tilde{\mathbf{S}}|$ is valid. When the capacity constraints of the storage areas are relatively tight considering product lots' inventory levels, there exist a small number of $\mathbf{r} \in \mathbf{R_i}$. Thus, in this case, $|\mathbf{S}|$ is significantly smaller than $|\tilde{\mathbf{S}}|$ and we can expect a shorter computation time with **S** compared to $\tilde{\mathbf{S}}$.

3.2.3. Set of actions

Action represents a decision maker selecting an alternative, based on the observed system state at the decision epoch. MDP-DBS expresses the action as the assignment of product lots to storage areas.

Let **a** indicate an action of a block stacking system representing assignment of product lots to storage areas and a^l be an element of **a** corresponding to product lot l. The system action **a** is defined as $\mathbf{a} = (a^1, a^2, \ldots, a^L)$ and the value of a^l indicates the index of a storage area. Let A^l_{il} be

the set of feasible actions of product lot l given i^l and $\mathbf{A_i}$ be the set of feasible actions of the block stacking system, given $\mathbf{i}$. Then, $\mathbf{A_i}$ is defined as follows:

$$\mathbf{A_i} = \left\{ \mathbf{a} \middle| a^l \in A^l_{i^l} \text{ where } 1 \le l \le L \text{ and } \sum_{\substack{1 \le l \le L \\ a^l = b}} n^l_{i^l,b} \le N_b \text{ for } b = 1, \dots, B \right\}. \quad (3.9)$$

where

$$n^l_{i^l,b} = \begin{cases} \lceil i^l/(d_b z^l) \rceil, & \text{if } i^l > 0 \\ \lceil Q^l/(d_b z^l) \rceil, & \text{if } i^l = 0 \end{cases}. \quad (3.10)$$

The inequality condition in (3.9) represents the capacity constraint of each storage area. Because the temporary storage area has infinite storage capacity ($N_B = \infty$), there exists at least one $\mathbf{a}$ satisfying the inequality condition in (3.9). Thus, the range of $\mathbf{A_i}$ is defined as follows:

$$1 \le |\mathbf{A_i}| \le B^L. \quad (3.11)$$

The set of feasible actions of the system $\mathbf{A_i}$ depends on $\mathbf{i} \in \mathbf{I}$ and $\mathbf{I}$ is stationary over a time horizon. Consequently, we assume $\mathbf{A_i}$ is stationary over a time horizon without loss of generality.

Observe the relationship between the state and the action of MDP-DBS. Let $\bar{\mathbf{r}}_t$ be the observed storage area state of the system at decision epoch t and let $\bar{a}_t$ indicate the selected action of the system at decision epoch t. The equation of $\bar{a}_t = \bar{r}_{t+1}$ is valid for all $t \in T$ because we assume no relocation for period t. Based on this observation, the possible state implies a feasible action is selected at the previous decision epoch.

3.2.4. State transition probability

In MDP-DBS, a probability mass function (PMF) of the system state at the next decision epoch can be derived from the observed system state and selected system action at the present decision epoch and the demand distribution of each product lot. When $\mathbf{s}$ and $\mathbf{a}$ indicate system state and

action at a decision epoch, respectively, let $\mathbf{s}_+$ be the system state at the next decision epoch, $\mathbf{i}_+$ be the inventory state of the system at the next decision epoch, and $\mathbf{r}_+$ be the storage area state of the system at the next decision epoch. Let i_+^l be the element of $\mathbf{i}_+$ corresponding to product lot l and r_+^l indicate the element $\mathbf{r}_+$ corresponding to product lot l.

We first derive the discrete probability distribution of r_+^l given a^l. Let $\bar{a}^l$ and $\bar{r}_+^l$ be the selected action of product lot l at a decision epoch and the observed storage area state of product lot l at the next decision epoch. As shown in Section 3.2.3, $\bar{r}_+^l$ is deterministically determined by $\bar{a}^l$ and the equation of $\bar{r}_+^l = \bar{a}^l$ is valid for given $\bar{a}^l$. Based on the observation, a PMF of r_+^l given a^l, $\Pr(r_+^l|a^l)$, is defined as follows:

$$\Pr(r_+^l|a^l) = \begin{cases} 1, & \text{if } r_+^l = a^l \\ 0, & \text{otherwise} \end{cases} \tag{3.12}$$

Next, consider a PMF of i_+^l given i^l. We define it using the fact D^l can be derived from i^l and i_+^l. Let $\bar{\imath}^l$, $\bar{D}^l$, and $\bar{\imath}_+^l$ be the observed inventory level state of product lot l at a decision epoch, the requested demand of product lot l during the following period, and the observed inventory level state of product lot l at the next decision epoch, respectively. As long as $\bar{\imath}^l \geq \bar{\imath}_+^l > 0$, $\bar{D}^l$ is simply computed by $\bar{\imath}^l$ - $\bar{\imath}_+^l$. When $\bar{\imath}^l$ is zero, $\bar{D}^l$ is calculated by Q^l - $\bar{\imath}_+^l$ because Q^l unit loads of product lot l is immediately replenished if $\bar{\imath}^l$ is zero. A zero inventory level at a decision epoch means the daily demand for the previous period is greater than or equal to the inventory level at the previous decision epoch. Thus, when $\bar{\imath}_+^l$ is zero, the range of $\bar{D}^l$ is defined as follows $\bar{\imath}^l \leq \bar{D}^l \leq D_{\max}^l$. Based on these observations, a PMF of i_+^l given i^l, $\Pr(i_+^l|i^l)$, is defined as follows when i^l is greater than zero:

$$\Pr(i_+^l|i^l) = \begin{cases} \Pr(D^l{=}i^l - i_+^l), & \text{if } i^l \geq i_+^l > 0 \\ \sum_{k=i^l}^{D_{\max}^l} \Pr(D^l{=}k), & \text{if } i_+^l = 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.13}$$

and when i^l is zero:

$$\Pr(i_+^l|i^l) = \begin{cases} \Pr(D^l{=}Q^l - i_+^l), & \text{if } Q^l \geq i_+^l > 0 \\ \sum_{k=Q^l}^{D_{\max}^l} \Pr(D^l{=}k), & \text{if } i_+^l = 0 \\ 0, & \text{otherwise} \end{cases} . \tag{3.14}$$

Let $p^l(s_+^l|s^l, a^l)$ be the PMF of s_+^l given s^l and a^l. It represents a probability of state transition from s^l to s_+^l for product lot l when taking action a^l and is defined as follows:

$$p^l(s_+^l|s^l, a^l) = \Pr(i_+^l|i^l) * \Pr(r_+^l|a^l). \tag{3.15}$$

Note s^l=(i^l, r^l) and s_+^l=(i_+^l, r_+^l). Thus, $p^l(s_+^l|s^l, a^l)$ can be expressed as follows:

$$p^l(s_+^l|s^l, a^l) = p^l\Big((i_+^l, r_+^l)\Big|(i^l, r^l), a^l\Big) = \begin{cases} \Pr(i_+^l|i^l), & \text{if } r_+^l{=}a^l \\ 0, & \text{if } r_+^l{\neq}a^l \end{cases}. \tag{3.16}$$

The sum of $p^l(s_+^l|s^l, a^l)$ over all s_+^l of $\mathbf{s}_+ \in \mathbf{S}$ equals one.

Let $P(\mathbf{s}_+|\mathbf{s}, \mathbf{a})$ be the PMF of $\mathbf{s}_+$ given $\mathbf{s}$ and $\mathbf{a}$. It expresses a probability of the system state transition from $\mathbf{s}$ to $\mathbf{s}_+$ when taking action $\mathbf{a}$. By the assumption of independent demand among product lots, $\Pr(i_+^l|i^l)$s of each product lot are statistically independent . Thus, $P(\mathbf{s}_+|\mathbf{s}, \mathbf{a})$ can be defined as a joint PMF of $p^l(s_+^l|s^l, a^l)$s where $1 \leq l \leq L$ as follows:

$$P(\mathbf{s}_+|\mathbf{s}, \mathbf{a}) = \prod_{1 \leq l \leq L} p^l(s_+^l|s^l, a^l) \tag{3.17}$$

The sum of $P(\mathbf{s}_+|\mathbf{s},\mathbf{a})$ over all $\mathbf{s}_+ \in \mathbf{S}$ is one as follows:

$$\begin{aligned}
\sum_{\mathbf{s}_+\in\mathbf{S}} P(\mathbf{s}_+|\mathbf{s},\mathbf{a}) &= \sum_{\mathbf{i}_+\in\mathbf{I}} \sum_{\mathbf{r}_+\in\mathbf{R_i}} \Pr(\mathbf{i}_+|\mathbf{i}) * \Pr(\mathbf{r}_+|\mathbf{a}) \\
&= \sum_{\mathbf{i}_+\in\mathbf{I}} \left(\Pr(\mathbf{i}_+|\mathbf{i}) \left(\sum_{\mathbf{r}_+\in\mathbf{R_i}} \Pr(\mathbf{r}_+|\mathbf{a}) \right) \right) = \sum_{\mathbf{i}_+\in\mathbf{I}} \Pr(\mathbf{i}_+|\mathbf{i}) \\
&= \sum_{\mathbf{i}_+\in\mathbf{I}} \left(\prod_{1\le l\le L} \Pr(i_+^l|i^l) \right) \qquad (3.18) \\
&= \sum_{i_+^1=0}^{Q^1} \sum_{i_+^2=0}^{Q^2} \cdots \sum_{i_+^L=0}^{Q^L} (\Pr(i_+^1|i^1) \Pr(i_+^2|i^2) \cdots \Pr(i_+^L|i^L)) \\
&= \sum_{k^1=0}^{D_{\max}^1} \sum_{k^2=0}^{D_{\max}^2} \cdots \sum_{k^L=0}^{D_{\max}^L} \Pr(D^1{=}k^1, \ldots, D^L{=}k^L) = 1
\end{aligned}$$

3.2.5. Reward

As a result of taking an action in the given state at the decision epoch, *reward* is given to the decision maker. In MDP-DBS, reward represents the operating cost incurred by selecting a daily DBS plan under the observed inventory levels and storage area of all product lots at the decision epoch. To avoid confusion, we have *cost* replace *reward* in MDP-DBS hereafter. In our research, the cost includes floor space cost, replenishment cost, retrieval cost, relocation cost. See APPENDIX A of the dissertation for details of the mathematical model of these costs.

In computing daily operating cost of the product lots, we assume the following scenario. At the decision epoch, a controller determines a DBS plan for the following period. If its inventory level is zero, the product lot is reordered and immediately replenished. If assigned storage area is different with the present storage area, the product lot is relocated instantly. During the next

period, unit loads of the product lots are retrieved by demand. APPENDIX A of the dissertation provides the daily operating cost model of a single product lot formulated as follows:

$$OC = FS + ST + RT + BT, \tag{3.19}$$

where

OC daily operating cost of a single product lot

FS floor space cost of a single product lot

ST replenishment cost of a single product lot

RT retrieval cost of a single product lot

BT relocation cost of a single product lot.

Costs can be easily redefined as a function of the product lot's state and action. We represent the cost functions using the following notations:

$FS(i^l, a^l)$ floor space cost function of inventory level state i^l and action a^l of product lot l

$ST(i^l, a^l)$ replenishment cost function of inventory level state i^l and action a^l or product lot l

$RT(i^l, a^l, D^l)$ retrieval cost function of inventory level state i^l, action a^l, and daily demand D^l of product lot l

$BT(i^l, r^l, a^l)$ relocation cost function of inventory level state i^l, storage area state r^l, and action a^l of product lot l.

Note i^l equaling zero means replenishment of production lot l; thus, when i^l is zero, it is assumed $i^l = Q^l$ in computing costs.

Retrieval cost is incurred to withdraw unit loads of product lot l for business hours the next day; thus, its calculation is based on the number of unit loads retrieved. Because we assume stochastic demand, exact daily demand of the product lot is unknown at the decision epoch.

Consequently, the retrieval cost is estimated as the expected value based on possible daily demands and their probabilities. The expected retrieval travel cost, $E[R^l(i^l, a^l)]$, is computed by

$$E[R^l(i^l, a^l)] = \sum_{k=0}^{D^l_{\max}} \Pr(D^l{=}k) * R^l(i^l, a^l, \min(i^l, k)). \tag{3.20}$$

Consider the temporary storage area. It must be guaranteed that only when all unit loads of product lots cannot be stored within regular storage area, the smallest number of unit loads are stored in the temporary storage area. In the MDP model, we avoid assigning a product lot to the temporary storage area unnecessarily and encourage relocating a product lot stored in the temporary storage area to the permanent storage area. Therefore, a penalty cost is imposed when a product lot is stored in the temporary storage area; a zero relocation cost is charged when a product lot is relocated from the temporary storage area to the regular storage area.

Let $c^l(s^l, a^l)$ be the cost incurred by product lot l over a single period as the result of taking action a^l in state s^l. It is computed as follows:

$$c^l(s^l, a^l) = \begin{cases} FS^l(i^l, a^l) + ST^l(i^l, a^l) + E[R^l(i^l, a^l)] + BT^l(i^l, r^l, a^l), & r^l \neq b^{\mathrm{T}} \text{ and } a^l = b^{\mathrm{T}} \\ FS^l(i^l, a^l) + ST^l(i^l, a^l) + E[R^l(i^l, a^l)], & r^l = b^{\mathrm{T}} \\ \psi \max\limits_{a^l}\{FS^l(i^l, a^l) + ST^l(i^l, a^l) + E[R^l(i^l, a^l)] + BT^l(i^l, r^l, a^l)\}, & a^l = b^{\mathrm{T}} \end{cases} \tag{3.21}$$

where ψ is the penalty factor greater than one. Penalty cost when a^l is b^{T} is computed by multiplying ψ and the maximum expected operating cost if product lot l is stored in one of the permanent storage areas.

Let $C(\mathbf{s}, \mathbf{a})$ be the daily operating cost incurred by all product lots when taking system action $\mathbf{a} \in \mathbf{A_i}$ in system state $\mathbf{s}$. It is computed by

$$C(\mathbf{s}, \mathbf{a}) = \sum_{l \in L} c^l(\mathrm{s}^l, a^l). \tag{3.22}$$

4. Solution Procedure

In this section, we develop heuristics to solve practical-sized instances of MDP-DBS. Generally, MDP determines a *policy* which is a collection of *decision rules* prescribing which action should be selected in each state. An infinite-horizon MDP finds a stationary policy independent of time and a finite-horizon MDP generates a non-stationary policy dependent on time. When a MDP model involves a large state space and/or action space, phenomenon of the curse of dimensionality arises and consequently it should be difficult to establish a policy. Inequality (3.8) and (3.11) indicate, in MDP-DBS, the size of the state space and action space increases exponentially as the number of product lots increases. Therefore, MDP-DBS also suffers from the curse of dimensionality and requires heuristics to solve practical-sized instances.

In Section 4.1, we introduce a *value iteration* algorithm. It is a widely-known solution method, establishing a policy for the infinite-horizon MDP. In Section 4.2, we develop a solution procedure based on an on-line approach for solving practical-sized instances of BSMPwRuSD. Instead of establishing a policy of MDP-DBS in advance, it instantly determines a good action for the observed state using a sampling technique.

4.1. Value iteration

Value iteration is a common algorithm for solving infinite-horizon MDP problem (Puterman, 2005). Basically, it recursively computes the optimality equation, also known as Bellman equation, until a termination condition is satisfied.

Figure 3.4 describes a procedure of the value iteration algorithm properly modified for MDP-DBS. In Steps 1 and 2, Equations (3.25) and (3.26) represent the optimality equation. The equation's first term indicates the immediate cost and second term represents the expected future

cost discounted to the present value. In the second term, λ is the discount factor such that $0 < \lambda \leq 1$. In Step 3 of the algorithm, $sp(v^n - v^{n-1})$ is computed as follows:

$$sp(v^n - v^{n-1}) = \max_{\mathbf{s}\in\mathbf{S}}[v^n(\mathbf{s}) - v^{n-1}(\mathbf{s})] - \min_{\mathbf{s}\in\mathbf{S}}[v^n(\mathbf{s}) - v^{n-1}(\mathbf{s})]. \quad (3.23)$$

In Step 4, letting π be a stationary policy of MDP-DBS, $\pi(\mathbf{s})$ represents the selected action given state $\mathbf{s}$. A stationary policy π established by the algorithm is referred to as ε-optimal policy.

Value iteration algorithm:

Step 1 Set $n = 1$ and specify convergence estimator $\varepsilon > 0$

Define $v^0(\mathbf{s})$ and $v^1(\mathbf{s})$ for each $\mathbf{s} \in \mathbf{S}$ as follows:

$$v^0 = \inf_{\mathbf{a}\in\mathbf{A}_i} C(\mathbf{s}, \mathbf{a}) \quad (3.24)$$

$$v^1(\mathbf{s}) = \min_{\mathbf{a}\in\mathbf{A}_i}\left\{C(\mathbf{s}, \mathbf{a}) + \sum_{\mathbf{s}_+\in\mathbf{S}} \lambda P(\mathbf{s}_+|\mathbf{s}, \mathbf{a})v^0(\mathbf{s}_+)\right\} \quad (3.25)$$

Step 2 For each $\mathbf{s} \in \mathbf{S}$, compute $v^{n+1}(\mathbf{s})$ by

$$v^{n+1}(\mathbf{s}) = \min_{\mathbf{a}\in\mathbf{A}_i}\left\{C(\mathbf{s}, \mathbf{a}) + \sum_{\mathbf{s}_+\in\mathbf{S}} \lambda P(\mathbf{s}_+|\mathbf{s}, \mathbf{a})v^n(\mathbf{s}_+)\right\} \quad (3.26)$$

Step 3 If $sp(v^n - v^{n-1}) - sp(v^{n+1} - v^n) < \varepsilon$, go to Step 4.

Otherwise, increase n by 1 and return to Step 2.

Step 4 For each $\mathbf{s} \in \mathbf{S}$, choose

$$\pi(\mathbf{s}) \in \operatorname*{argmin}_{\mathbf{a}\in\mathbf{A}_i}\left\{C(\mathbf{s}, \mathbf{a}) + \sum_{\mathbf{s}_+\in\mathbf{S}} \lambda P(\mathbf{s}_+|\mathbf{s}, \mathbf{a})v^n(\mathbf{s}_+)\right\} \quad (3.27)$$

and stop.

Figure 3.4: Value iteration algorithm for MDP-DBS

4.2. Instant determination heuristics

In theory, the value iteration algorithm of Section 4.1 can establish an optimal stationary policy of MDP-DBS but we face computational difficulties caused by a large state space and action space when solving practical-sized instances.

An approach to address the large state space is to simplify the MDP model via some approximation techniques such as state aggregation (Boutilier *et al*. 1999), value function approximation (Powell, 2011), randomization (Rust, 1997), and so on. To tackle the MDP problem with large state space, instead of constructing an optimal policy, Peret and Garcia (2004) and Nicol and Chades (2011) introduced the strategy of finding an optimal action for a single current state based on future states sampled by simulating the system. Their method can be considered an on-line approach compared to the methods predetermining an action for every state in an off-line manner.

To solve practical-sized instances of BSMPwRuSD, we develop a solution procedure referred to as Instant Determination Heuristic (IDH). To address a large state space of MDP-DBS, IDH adopts an on-line approach of instantly determining an action for the observed state at a decision epoch. Therefore, IDH avoids searching every state to determine an optimal policy. Additionally, to alleviate the computational difficulty caused by a large action space, IDH formulates the instant action determination problem as a General Assignment Problem (GAP). Using an advanced solution procedure like branch-and-bound, IDH avoids enumerating all possible actions in finding a solution.

Figure 3.5 depicts the flow of IDH. At first, at a decision epoch, product lots' inventory levels and current storage areas are observed. Then, IDH samples product lots' daily demands over a specified horizon using stochastic simulation and computes product lots' expected inventory levels from the sampled daily demand. Next, IDH calculates product lots' approximate minimum future costs using sampled daily demands and computed inventory levels. Given product lots' approximate minimum future costs, IDH formulates and solves the GAP of the instant action determination problem. Finally, IDH obtains the assignment of product lots to a storage area for the decision epoch from the solution of the GAP.

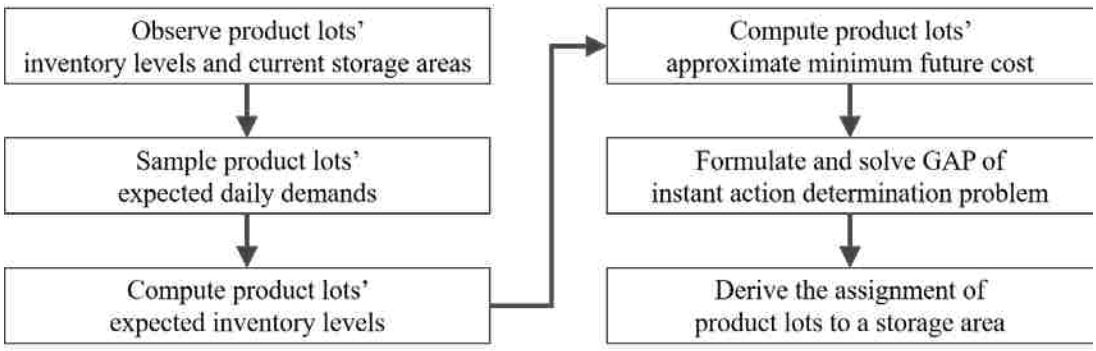

Figure 3.5: Flow of Instant Determination Heuristic (IDH)

In Section 4.2.1, we look at the GAP of the instance action determination problem. Section 4.2.2 describes a procedure of sampling expected daily demands and computing expected inventory levels over a specified horizon. Section 4.2.3 introduces different ways of calculating an approximate minimum future cost of a product lot. We develop an algorithmic expression of IDH in Section 4.2.4 and investigate the performance of IDH in Section 4.2.5.

4.2.1. Instant action determination problem

Let $\bar{\mathbf{s}}_{\hat{t}}$ be the system state observed at decision epoch $\hat{t}$ and $\bar{\mathbf{i}}_{\hat{t}}$ and $\bar{\mathbf{r}}_{\hat{t}}$ be the element of $\bar{\mathbf{s}}_{\hat{t}}$. The element of $\bar{\mathbf{i}}_{\hat{t}}$ corresponding to product lot l is referred to as $\bar{\imath}_{\hat{t}}^{l}$ and the element of $\bar{\mathbf{r}}_{\hat{t}}$ corresponding to product lot l is referred to as $\bar{r}_{\hat{t}}^{l}$. In IDH, given state $\bar{\mathbf{s}}_{\hat{t}}$, the instant action determination problem is formulated as a GAP as follows:

IDH-GAP:

$$\min \sum_{l=1}^{L} \sum_{b=1}^{B} \left(c^{l}\left(\bar{s}_{\hat{t}}^{l}, b\right) + f^{l}\left(\bar{s}_{\hat{t}}^{l}, b\right) \right) x_{b}^{l} \tag{3.28}$$

subject to

$$\sum_{l=1}^{L} n_{\bar{\imath}_{\hat{t}}^{l}, b}^{l} x_{b}^{l} \leq N_{b}, \qquad b = 1, \ldots, B \tag{3.29}$$

$$\sum_{b=1}^{B} x_b^l = 1, \qquad l = 1, \dots, L \tag{3.30}$$

$$x_b^l \in \{0,1\}, \qquad b = 1, \dots, B, l = 1, \dots, L \tag{3.31}$$

Decision variable x_b^l indicates product lot l is assigned to a storage area b if x_b^l is one or not assigned if x_b^l is zero. Constraints (3.29) force the number of required row positions for the product lots assigned to storage area b to be less than or equal to the capacity of storage area b. Constraints (3.30) guarantee a product lot is assigned to only one storage area. Constraints (3.31) prohibit lot splitting by requiring the x-variables to take on binary values.

The first term in the objective function, $c^l(\bar{s}_{\hat{t}}^l, b)$ as introduced in Section 3.2.5, indicates the expected daily operating cost at decision epoch $\hat{t}$ when product lot l is assigned to storage area b given system state $\bar{s}_{\hat{t}}^l$. When assigning product lot l to storage area b given $\bar{s}_{\hat{t}}^l$ is infeasible because of the capacity constraint of storage area b, $c^l(\bar{s}_{\hat{t}}^l, b)$ is set equal to a very large value compared to the general daily operating cost. The second term $f^l(\bar{s}_{\hat{t}}^l, b)$ represents the expected future operating cost over a time horizon resulting from the assignment of product lot l to a storage area b given state $\bar{s}_{\hat{t}}^l$. By including the future cost element in the objective function, we pursue the global optimization over a time horizon rather than the local optimization at a given day. We introduce three different computational strategies for estimating $f^l(\bar{s}_{\hat{t}}^l, b)$ in the following sections; two are based on a stochastic simulation.

4.2.2. Sampling daily demand and inventory level

At decision epoch $\hat{t}$, in estimating $f^l(s^l, b)$ over SL days (i.e., sample length), we use randomly generated sample paths of D^l and i^l instead of all possible sample paths. Note *sample path* indicates the sequence of realized random variables in this research. A sample path of D^l is generated from a stochastic simulation and a sample path of i^l is computed from a sample path of D^l.

Let D^l be the random daily demand of product lot l and $\widetilde{D}_t^l$ be a realized D^l at day t in a stochastic simulation. The sample path of D^l from day $\hat{t}$ to day $\hat{t}+SL$ is referred to as $\mathbb{D}_{\hat{t}}^l$ and defined as follows:

$$\mathbb{D}_{\hat{t}}^l = \left(\widetilde{D}_{\hat{t}}^l, \widetilde{D}_{\hat{t}+1}^l, \dots, \widetilde{D}_{\hat{t}+SL}^l\right). \tag{3.32}$$

Let $\mathbb{D}_{\hat{t}}^l(j)$ indicate the element of $\mathbb{D}_{\hat{t}}^l$ corresponding to day $\hat{t}+j$ for j=0,1,…,SL. Note the length of $\mathbb{D}_{\hat{t}}^l$ is SL+1. Particularly, the sample path $\mathbb{D}_{\hat{t}}^l$ generated with a fixed $\widetilde{D}_{\hat{t}}^l = k$ is referred to as $\mathbb{D}_{\hat{t}|k}^l$ and defined as follows:

$$\mathbb{D}_{\hat{t}|k}^l = \left(k, \widetilde{D}_{\hat{t}+1}^l, \dots, \widetilde{D}_{\hat{t}+SL}^l\right), \quad k=0,1,\dots,D_{\max}^l \tag{3.33}$$

Because we generate multiple $\mathbb{D}_{\hat{t}|k}^l$s for each k in the sampling procedure, $\mathbb{D}_{\hat{t}|k}^l$ is indexed with σ as $\mathbb{D}_{\hat{t}|k|\sigma}^l$. When SN (i.e., sample number) indicates the number of $\mathbb{D}_{\hat{t}|k}^l$ for each k, the number of sample paths generated for D^l is computed by $SN * (D_{\max}^l+1)$.

Let i^l be the random inventory level of product lot l and $\tilde{\imath}_t^l$ be a computed i^l based on $\widetilde{D}_{\hat{t}}^l$. The sample path of i^l from day $\hat{t}$ to day $\hat{t}+SL$ is referred to as $\mathbb{i}_{\hat{t}}^l$ and defined as follows:

$$\mathbb{i}_{\hat{t}}^l = \left(\bar{\imath}_{\hat{t}}^l, \tilde{\imath}_{\hat{t}+1}^l, \dots, \tilde{\imath}_{\hat{t}+SL}^l\right) \tag{3.34}$$

Note $\bar{\imath}_{\hat{t}}^l$ is the inventory level of product lot l observed at decision epoch $\hat{t}$. Let $\mathbb{i}_{\hat{t}}^l(j)$ indicate the element of $\mathbb{i}_{\hat{t}}^l$ corresponding to day $\hat{t}+j$ for j=0,1,…,SL. Note the length of $\mathbb{i}_{\hat{t}}^l$ is SL+1. Specifically, the sample path $\mathbb{i}_{\hat{t}}^l$ derived from $\mathbb{D}_{\hat{t}|k|\sigma}^l$ is referred to as $\mathbb{i}_{\hat{t}|k|\sigma}^l$. Given $\mathbb{D}_{\hat{t}|k|\sigma}^l$, with $\mathbb{i}_{\hat{t}|k|\sigma}^l(0) = \bar{\imath}_{\hat{t}}^l$, $\mathbb{i}_{\hat{t}|k|\sigma}^l$ is computed as follows:

$$\mathbb{i}_{\hat{t}|k|\sigma}^l(j) = \begin{cases} \max\left(0, \mathbb{i}_{\hat{t}|k|\sigma}^l(j-1) - \mathbb{D}_{\hat{t}|k|\sigma}^l(j-1)\right), & \text{if } \mathbb{i}_{\hat{t}|k|\sigma}^l(j-1) > 0 \\ \max\left(0, Q^l - \mathbb{D}_{\hat{t}|k|\sigma}^l(j-1)\right), & \text{if } \mathbb{i}_{\hat{t}|k|\sigma}^l(j-1) = 0 \end{cases}, \text{ for } j = 1, \dots, SL. \tag{3.35}$$

For example, consider a product lot l with Q^l equal to five and $D^l_{\max}$ equal to two and assume SL is equal to five and SN is equal to three. At decision epoch 3, $\mathbb{D}^l_{\hat{t}|k|\sigma}$ and $\mathbb{i}^l_{\hat{t}|k|\sigma}$ are defined as shown in Table 3.4 when the observed inventory level of product lot l, $\bar{\imath}^l_{\hat{t}=3}$, is equal to three.

In the rows of Table 3.4 corresponding to k = 0 and σ = 3, $\mathbb{i}^l_{3|0|3}$ = (3, 3, 1, 0, 3, 3) and $\mathbb{D}^l_{3|0|3}$= (0, 2, 2, 2, 0, 1). The value of $\mathbb{i}^l_{3|0|3}(0)$ is set equal to three because $\bar{\imath}^l_3$ is equal to three. The value of $\mathbb{i}^l_{3|0|3}(1)$ equal to three is computed as follows: $\mathbb{i}^l_{3|0|3}(0)$ minus $\mathbb{D}^l_{3|0|3}(0)$ or 3 – 0 = 3. The value of $\mathbb{i}^l_{3|0|3}(3)$ is set equal to zero by max(0, -1) because $\mathbb{i}^l_{3|0|3}(2)$ minus $\mathbb{D}^l_{3|0|3}(2)$ is negative as follows 1 – 2 = –1. The value of $\mathbb{i}^l_{3|0|3}(4)$ equal to three is computed as follows: Q^l minus $\mathbb{D}^l_{3|0|3}(3)$ or 5 – 2 = 3 because $\mathbb{i}^l_{3|0|3}(3)$ is zero.

Table 3.4: Example of sample paths of inventory level and daily demand

Parameters:	Q^l: 5	$D^l_{\max}$: 2	$\bar{\imath}^l_3$: 3		SL: 5		SN: 3	
Demand at day 3 (k)	σ	Sample path	j					
			0	1	2	3	4	5
0	1	$\mathbb{i}^l_{3\|0\|1}$	3	3	2	0	4	3
		$\mathbb{D}^l_{3\|0\|1}$	0	1	2	1	1	0
	2	$\mathbb{i}^l_{3\|0\|2}$	3	3	3	2	1	0
		$\mathbb{D}^l_{3\|0\|2}$	0	0	1	1	2	0
	3	$\mathbb{i}^l_{3\|0\|3}$	3	3	1	0	3	3
		$\mathbb{D}^l_{3\|0\|3}$	0	2	2	2	0	1
1	1	$\mathbb{i}^l_{3\|1\|1}$	3	2	2	0	4	4
		$\mathbb{D}^l_{3\|1\|1}$	1	0	2	1	0	2
	2	$\mathbb{i}^l_{3\|1\|2}$	3	2	1	0	4	2
		$\mathbb{D}^l_{3\|1\|2}$	1	1	1	1	2	0
	3	$\mathbb{i}^l_{3\|1\|3}$	3	2	0	4	3	2
		$\mathbb{D}^l_{3\|1\|3}$	1	2	1	1	1	1
2	1	$\mathbb{i}^l_{3\|2\|1}$	3	1	1	1	1	1
		$\mathbb{D}^l_{3\|2\|1}$	2	0	0	0	0	1
	2	$\mathbb{i}^l_{3\|2\|2}$	3	1	0	5	3	1
		$\mathbb{D}^l_{3\|2\|2}$	2	1	0	2	2	0
	3	$\mathbb{i}^l_{3\|2\|3}$	3	1	0	5	3	2
		$\mathbb{D}^l_{3\|2\|3}$	2	1	0	2	1	2

Let $[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$ be the aggregate sample path of $\mathbb{i}^l_{\hat{t}|k|\sigma}$ and $\mathbb{D}^l_{\hat{t}|k|\sigma}$ over the *SL*-days-horizon from day $\hat{t}$+1 to day $\hat{t}$+*SL*. It unites $\mathbb{i}^l_{\hat{t}|k|\sigma}(j)$ and $\mathbb{D}^l_{\hat{t}|k|\sigma}(j)$, j=1,…, *SL* and does not include $\mathbb{i}^l_{\hat{t}|k|\sigma}(0)$ and $\mathbb{D}^l_{\hat{t}|k|\sigma}(0)$. Note that $\mathbb{i}^l_{\hat{t}|k|\sigma}(0)$ and $\mathbb{D}^l_{\hat{t}|k|\sigma}(0)$ represent a possible scenario of the system at decision epoch $\hat{t}$ and are not related to the expected future operating cost.

Let $c\left([\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}\right)$ indicate the operating cost on $[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$. Then, $f^l\left(\bar{s}^l_{\hat{t}}, b\right)$ is estimated as a weighted average of $c\left([\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}\right)$ as follows:

$$f^l\left(\bar{s}^l_{\hat{t}}, b\right) = \sum_{k=0}^{D^l_{\max}} \sum_{\sigma=1}^{SN} \left(\frac{\Pr(D^l=k)}{SN} c\left([\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}\right) \right), \tag{3.36}$$

where $\Pr(D^l=k)/SN$ works as the weight. Note the sum of the weights is one as follows:

$$\sum_{k=0}^{D^l_{\max}} \sum_{\sigma=1}^{SN} \frac{\Pr(D^l=k)}{SN} = \sum_{k=0}^{D^l_{\max}} \left(\Pr(D^l=k) \sum_{\sigma=1}^{SN} \frac{1}{SN} \right) = \sum_{k=0}^{D^l_{\max}} \Pr(D^l=k) = 1. \tag{3.37}$$

4.2.3. Approximate minimum future cost

In computing $c\left([\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}\right)$, we employ three different strategies: Myopic Policy (MP) strategy, Block Stacking Single Product with Relocation under Stochastic Demand (BSSPwRuSD) strategy, and Block Stacking Single Product with Relocation under Deterministic Demand (BSSPwRuSD) strategy.

MP strategy

Unlike other strategies, the MP strategy only considers the immediate cost in finding an optimal solution of IDH-GAP. Let $f^l\left(\bar{s}^l_{\hat{t}}, b\right)^{\mathrm{M}}$ be the expected future cost estimated by the MP strategy. The value of $f^l\left(\bar{s}^l_{\hat{t}}, b\right)^{\mathrm{M}}$ is zero, i.e.,

$$f^l\left(\bar{s}_{\hat{t}}^l, b\right)^{\mathrm{M}} = 0 \tag{3.38}$$

We refer to IDH-GAP with $f^l\left(\bar{s}_{\hat{t}}^l, b\right)^{\mathrm{M}}$ as IDH-GAP$^{\mathrm{M}}$.

BSSPwRuSD strategy

The BSSPwRuSD strategy uses the solution obtained by optimizing DBS of a single product lot under stochastic demand, called the BSSPwRuDD problem. An MDP-DBS of a single product lot l is framed using a MDP quintuple as $\{T, S^l, A_{i^l}^l, p(\cdot\,|s^l, a^l), c(s^l, a^l)\}$; an optimal policy for product lot l can be easily established by value iteration or policy iteration. For more details of the MDP-DBS of a single product lot and the BSSPwRuSD problem, see Lee *et al*. (2016) or APENDDIX B of the dissertation.

In computing $c\left([\mathbb{i}, \mathbb{D}]_{\hat{t}+1|k|\sigma}^l\right)$ at decision epoch $\hat{t}$, the BSSPwRuSD strategy defines a sequence of a^ls over the SL-days-horizon from day $\hat{t}$+1 to day $\hat{t}$+SL using the policy of product lot l determined by solving MDP-DBS of product lot l. Let π^l be the policy of product lot l and $\pi^l(i^l, r^l)$ represent the predetermined action in state (i^l, r^l). The BSSPwRuSD strategy computes $c\left([\mathbb{i}, \mathbb{D}]_{\hat{t}+1|k|\sigma}^l\right)$ as follows:

$$c\left([\mathbb{i}, \mathbb{D}]_{\hat{t}+1|k|\sigma}^l\right) = \sum_{j=\hat{t}+1}^{\hat{t}+SL} \lambda^{j-\hat{t}}\, \tilde{c}^l\left(\mathbb{i}_{\hat{t}|k|\sigma}^l(j), \bar{r}_j^l, \pi^l\left(\mathbb{i}_{\hat{t}|k|\sigma}^l(j), \bar{r}_j^l\right), \mathbb{D}_{\hat{t}|k|\sigma}^l(j)\right), \tag{3.39}$$

where

$$\bar{r}_j^l = \begin{cases} b, & \text{for } j{=}t{+}1 \\ \pi^l\left(\mathbb{i}_{\hat{t}|k|\sigma}^l(j-1), \bar{r}_{j-1}^l\right), & \text{for } j{=}t{+}2, \ldots, t{+}SL. \end{cases} \tag{3.40}$$

As a variant of (3.21), cost function $\tilde{c}^l(i^l, r^l, a^l, D^l)$ of (3.39) is defined as follows:

$$\tilde{c}^l(i^l, r^l, a^l, D^l)$$

$$= \begin{cases} FS^l(i^l, a^l) + ST^l(i^l, a^l) + RT\left(i^l, a^l, D^l\right) + BT^l(i^l, r^l, a^l), & r^l \neq b^{\mathrm{T}} \text{ and } a^l = b^{\mathrm{T}} \\ FS^l(i^l, a^l) + ST^l(i^l, a^l) + RT\left(i^l, a^l, D^l\right), & r^l = b^{\mathrm{T}} \\ \psi \max\limits_{a^l}\left\{FS^l(i^l, a^l) + ST^l(i^l, a^l) + RT\left(i^l, a^l, D^l\right) + BT^l(i^l, r^l, a^l)\right\}, & a^l = b^{\mathrm{T}} \end{cases} . \quad (3.41)$$

Note in computing $c\left([\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}\right)$ to estimate $f^l\left(\bar{s}^l_{\hat{t}}, b\right)$, we assume product lot l is assigned to storage area b at decision epoch $\hat{t}$. Thus, in (3.40), $\bar{r}^l_{\hat{t}+1}$ is set as b.

Let $c\left([\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}\right)^{\mathrm{S}}$ be the operating cost computed on sample $[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$ by the BSSPwRuSD strategy. Let $f^l(s^l, b)^{\mathrm{S}}$ indicate the expected future cost estimated using $c\left([\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}\right)^{\mathrm{S}}$ and IDH-GAP$^{\mathrm{S}}$ represent IDH-GAP with $f^l(s^l, b)^{\mathrm{S}}$. When the BSSPwRuSD strategy is chosen, the BSSPwRuSD problem is solved for all product lots to establish their policies before implementing IDH.

BSSPwRuDD strategy

The BSSPwRuDD strategy includes solving the problem of optimizing DBS of a single product lot under deterministic demand, called the BSSPwRuDD problem. The problem assumes product lot's exact inventory level and daily demand each day are known over a time horizon; a solution of the problem determines a product lot's DBS plan over a time horizon minimizing the total operating cost. CHAPTER 2 of the dissertation provides details of DBS under deterministic demand.

In the domain of $[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$, daily demand of product lot l is deterministic and consequently, the exact inventory levels of product lot l over the SL-days-horizon from day $\hat{t}$+1 to day $\hat{t}$+SL is known in advance. Hence, for given sample path $[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$, we can formulate the BSSPwRuDD problem, referred to as BSSPwRuDD-$[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$. Solving BSSPwRuDD-$[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$

determines a sequence of a^l over the SL-days-horizon minimizing total operating cost on sample path $[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$.

At decision epoch $\hat{t}$, BSSPwRuDD-$[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$ is formulated as a shortest path problem on a directed graph illustrated in Figure 3.6. It represents DBS of product lot l in the domain of $[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$.

In the graph of Figure 3.6, the nodes represent storage areas at each day over the SL-days-horizon. Storage area δ at day $\hat{t}+j$ is represented as node $jB+\delta$ where j = 0,…,SL and δ = 1,…,B. The node set, N, is defined as follows:

$$N = \{jB + \delta | j = 0, \dots, SL, \delta = 1, \dots, B\} \cup \{s, e\}.$$

where nodes s and e indicate the start node and end node, respectively. We refer to the set of nodes corresponding to day $\hat{t}+j$ as $N_{\hat{t}+j}$ and define it as follows:

$$N_{\hat{t}+j} = \{jB + \delta | \delta = 1, \dots, B\}.$$

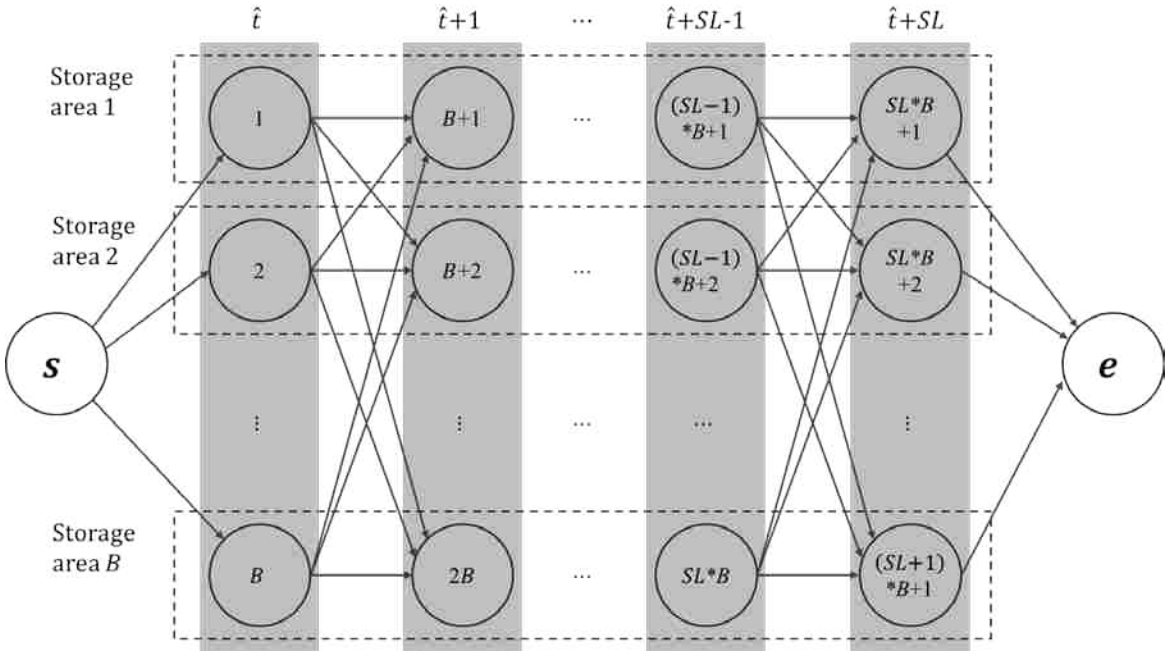

Figure 3.6: Directed graph of the aggregate sample path from day $\hat{t}$ to day $\hat{t}$+SL

A directed arc indicates an action in a given state of the storage area. The arc originating from node $(j\text{-}1)B+\gamma$ to node $jB+\delta$ denotes an action $a^l=\delta$ in given $r^l=\gamma$. From a node $jB+\delta$, B arcs emanate to all nodes in $N_{\hat{t}+j+1}$ where $j=0, \dots, SL\text{-}1$ and $\delta=1, \dots, B$. The set of arcs, A, is defined as follows:

$$A = \{(jB+\gamma, (j+1)R+\delta) | j = 0, \dots, SL-1, \gamma = 1, \dots, B, \delta = 1, \dots, B\}$$
$$\cup\{(s, \delta) | \delta = 1, \dots, B\} \cup \{(SL * B + \delta, e) | \delta = 1, \dots, B\}.$$

Let $\bar{A}_i$ and $\underline{A}_i$ respectively indicate the set of arcs incident to node i and the set of arcs emanating from node i.

The cost of the arc $\big((j\text{-}1)B+\gamma,\ jR+\delta\big)$ for product lot l is referred to as $c^l_{(j\text{-}1)B+\gamma, jR+\delta}$ and denotes the operating cost incurred by product lot l at day $\hat{t}+j$ when action a^l is δ, inventory level state i^l is $\mathbb{i}^l_{\hat{t}|k|\sigma}(j)$, storage area state r^l is γ, and daily demand D^l is $\mathbb{D}^l_{\hat{t}|k|\sigma}(j)$. Using the cost function $\tilde{c}^l(i^l, r^l, a^l, D^l)$ of (3.41), $c^l_{(j\text{-}1)B+\gamma, jR+\delta}$ is defined as follows:

$$c^l_{(j\text{-}1)B+\gamma, jR+\delta} = \begin{cases} \lambda^j \tilde{c}^l\left(\mathbb{i}^l_{\hat{t}|k|\sigma}(j), \gamma, \delta, \mathbb{D}^l_{\hat{t}|k|\sigma}(j)\right), & \text{if } n^l_{\mathbb{i}^l_{\hat{t}|k|\sigma}(j), \delta} \le N_\delta \\ M, & \text{otherwise} \end{cases}, \tag{3.42}$$

where $j = 1, \dots, SL,\ \gamma = 1, \dots, B,\ \delta = ,1 \dots, B$ and M represents a very large value compared to the common daily operating cost of a single product lot. Note $n^l_{\mathbb{i}^l_{\hat{t}|k|\sigma}(j), \delta}$ indicates the required number of row positions for storing $\mathbb{i}^l_{\hat{t}|k|\sigma}(j)$ unit loads of product lot l in storage area δ (or d_δ-deep storage area). Cost of an arc is set as M if the arc indicates the infeasible action and consequently, the arc is excluded from the optimal solution of BSSPwRuDD-$[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$. Costs of arcs connected with node s and e are set as zero.

On the developed directed graph, BSSPwRuDD-$[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$ is formulated as a shortest path problem as follows:

BSSPwRuDD$\left([\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}\right)$:

$$\min \sum_{(i,j)\in A} c^l_{ij} x^l_{ij} \tag{3.43}$$

subject to

$$\sum_{(i,j)\in \overline{\boldsymbol{A}}_j} x^l_{ij} - \sum_{(j,k)\in \underline{\boldsymbol{A}}_j} x^l_{jk} = \begin{cases} -1, & j = s \\ 0, & j \in N \setminus \{s, e\} \\ 1, & j = e \end{cases} \tag{3.44}$$

$$x^l_{s,b} = 1 \tag{3.45}$$

$$x^l_{ij} \in \{0,1\}, \quad \forall (i,j) \in A \tag{3.46}$$

Decision variable x^l_{ij} is equal to one if arc (i,j) is selected in the solution, or zero otherwise. Constraint (3.44) is a general flow balance constraint making product lot l's supply and demand identical at each node. It guarantees, in a DBS plan for product lot l, only one storage area is chosen for product lot l each day. Constraint (3.45) forces arc (s,b) be chosen in the solution of BSSPwRuDD-$[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$. Note in computing $c\left([\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}\right)$ to estimate $f^l\left(\bar{s}^l_{\hat{t}}, b\right)$, we assume product lot l is assigned to storage area b at decision epoch $\hat{t}$. Constraint (3.46) prohibits lot splitting by requiring the x-variables to take on binary values.

In estimating $f^l\left(\bar{s}^l_{\hat{t}}, b\right)$ at decision epoch $\hat{t}$, the BSSPwRuDD strategy sets $c\left([\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}\right)$ as the optimal objective function value of BSSPwRuDD-$[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$. Let $c\left([\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}\right)^{\mathrm{D}}$ be the operating cost computed on sample $[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$ by the BSSPwRuDD strategy. Let $f^l\left(\bar{s}^l_{\hat{t}}, b\right)^{\mathrm{D}}$ be the expected future cost estimated using $c\left([\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}\right)^{\mathrm{D}}$ and IDH-GAP$^{\mathrm{D}}$ represent IDH-GAP with $f^l(s^l, b)^{\mathrm{D}}$. In implementing IDH with the BSSPwRuDD strategy, BSSPwRuDD-$[\mathbb{i}, \mathbb{D}]^l_{\hat{t}+1|k|\sigma}$ is solved $(1 + D^l_{\max}) * SN$ times in estimating each $f^l\left(\bar{s}^l_{\hat{t}}, b\right)^{\mathrm{D}}$ and consequently, $L * B * (1 + D^l_{\max}) * SN$ times in total.

4.2.4. Algorithmic expression of IDH

In this section, we look at an pseudo-code of IDH implemented at decision epoch $\hat{t}$. The notation of Table 3.5 is used to simplify the pseudo-code.

Table 3.5: Notations of the pseudo-code of the IDH algorithm

Notation	*Description*
CS	acronym of Computational Strategy
M, S, D	index of the Myopic strategy (M), the BSSPwRuSD strategy (S), and the BSSPwRuDD strategy (D)
D_{ave}^{l}	average daily demand of product lot l
$\text{Rand}\left(D_{\text{ave}}^{l}\right)$	random number generated according to the probability distribution of daily demand of product lot l whose average daily demand is D_{ave}^{l}, $0 \leq \text{Rand}\left(D_{\text{ave}}^{l}\right) \leq D_{\max}^{l}$
$\bar{\mathbf{a}}_{\hat{t}}$	selected system action at decision epoch $\hat{t}$
$\bar{a}_{\hat{t}}^{l}$	elements of $\bar{\mathbf{a}}_{\hat{t}}$ corresponding to $\hat{t}$

Figure 3.7 describes the pseudo-code of IDH. Line (1) sets $\bar{\mathbf{s}}_{\hat{t}}$, $\bar{\mathbf{i}}_{\hat{t}}$, and $\bar{\mathbf{r}}_{\hat{t}}$ based on the observed system state at decision epoch $\hat{t}$ and Line (2) computes immediate operating costs for product lot l, l = 1,…,L and alternative storage area b, b=1,…,B. Lines from (3) and (5) are the procedure when the Myopic strategy is selected. Line (4) sets the expected future costs of all combinations of product lot l and the alternative storage area b as zero and Line (5) solves IDH-GAP with the expected costs set by Line (4). Lines from (6) to (27) are the procedure when either the BSSPwRuSD strategy or the BSSPwRuDD strategy is selected. For a single combination of product lot l and alternative storage area b, Lines from (11) to (20) generate a single sample path and Line (21) computes operating cost on the single sample path. Line (24) computes expected future cost based on operating costs of sample paths computed by Lines from (9) to (23). Line (27) solves IDH-GAP with the expected future costs computed for all combinations of product lot l and alternative storage area b by Lines from (7) to (26). Line (29)

IDH Algorithm:

Set $\hat{t}$ as the present decision epoch and define $\bar{\mathbf{s}}_{\hat{t}}$, $\bar{\mathbf{i}}_{\hat{t}}$, and $\bar{\mathbf{r}}_{\hat{t}}$ (1)

Compute $c^l\left(\bar{s}_{\hat{t}}^l, b\right)$ for $l = 1, \ldots, L$ and $b = 1, \ldots, B$ (2)

IF *CS* is M (3)

Set $f^l\left(\bar{s}_{\hat{t}}^l, b\right)^{\mathrm{M}} = 0$ for $l = 1, \ldots, L$ and $b = 1, \ldots, B$ (4)

Solve IDH-GAP$^{\mathrm{M}}$ (5)

ELSE IF *CS* is S or D (6)

FOR l = 1 to L (7)

FOR b = 1 to B (8)

FOR k = 0 to $D_{\max}^l$ (9)

FOR σ = 0 to SN (10)

Do $\mathbb{i}_{\hat{t}|k|\sigma}^l(0) \leftarrow \bar{\imath}_{\hat{t}}^l$ (11)

FOR j = 0 to SL-1 (12)

Do $\mathbb{D}_{\hat{t}|k|\sigma}^l(j) \leftarrow \mathrm{Rand}(D_{\mathrm{average}}^l)$ (13)

IF $\mathbb{i}_{\hat{t}|k|\sigma}^l(j) > 0$ (14)

Do $\mathbb{i}_{\hat{t}|k|\sigma}^l(j{+}1) \leftarrow \max\left(0, \mathbb{i}_{\hat{t}|k|\sigma}^l(j) - \mathbb{D}_{\hat{t}|k|\sigma}^l(j)\right)$ (15)

ELSE IF $\mathbb{i}_{\hat{t}|k|\sigma}^l(j) = 0$ (16)

Do $\mathbb{i}_{\hat{t}|k|\sigma}^l(j{+}1) \leftarrow \max\left(0, Q^l - \mathbb{D}_{\hat{t}|k|\sigma}^l(j)\right)$ (17)

END IF (18)

NEXT FOR (19)

$\mathbb{D}_{\hat{t}|k|\sigma}^l(SL) \leftarrow \mathrm{Rand}(D_{\mathrm{average}}^l)$ (20)

Compute $c\left([\mathbb{i}, \mathbb{D}]_{\hat{t}+1|k|\sigma}^l\right)^{CS}$ (21)

NEXT FOR (22)

NEXT FOR (23)

Do

$$f^l\left(\bar{s}_{\hat{t}}^l, b\right)^{CS} \leftarrow \sum_{k=0}^{D_{\max}^l} \sum_{\sigma=1}^{SN} \left(\frac{\Pr(D^l{=}k)}{SN} c\left([\mathbb{i}, \mathbb{D}]_{\hat{t}+1|k|\sigma}^l\right)^{CS} \right) \quad (24)$$

NEXT FOR (25)

NEXT FOR (26)

Solve IDH-GAP^{CS} (27)

END IF (28)

Do

$$\bar{a}_{\hat{t}}^l = \underset{b}{\mathrm{argmax}}\left\{x_b^l \middle| b = 1, \ldots, B\right\} \text{ for } l = 1, \ldots, L \quad (29)$$

Return $\bar{\mathbf{a}}_{\hat{t}} = \left(\bar{a}_{\hat{t}}^1, \bar{a}_{\hat{t}}^2, \ldots \bar{a}_{\hat{t}}^L\right)$ (30)

Figure 3.7: Pseudo-code of IDH algorithm

defines the selected action for each product lot based on the solution of IDH-GAP. Note in the solution of IDH-GAP, $x_b^l = 1$ indicates product lot l is assigned to storage area b; by Constraints (3.30) of IDH-GAP, for product lot l, only one decision variable is equal to one and the others are equal to zero. Therefore, the right hand side of the equation in Line (29) returns the index of the selected storage area for product lot l.

4.2.5. Performance of IDH

In this section, benchmarking the performance of the optimal policy of MDP-DBS, we define IDH's performance mathematically referring to Chang *et al*. (2013)'s theorem about the approximate rolling horizon control's performance. The approximate rolling horizon control estimates the minimum future cost of a product lot using a sampling technique like IDH but, unlike IDH, predetermines an action for every state in the off-line fashion. To facilitate defining IDH's performance mathematically, consider the notation of Table 3.6.

Table 3.6: Notations for defining performance of IDH

Notation	*Description*
MDP-DBS_∞	infinite horizon MDP-DBS
MDP-DBS_{SL}	finite horizon MDP-DBS over period from day 0 to SL
$\pi^*, \pi^*(\mathbf{s})$	stationary optimal policy of MDP-DBS_∞, predetermined action for state $\mathbf{s}$ in π^*
$\pi^{\text{fh}}, \pi_t^{\text{fh}}(\mathbf{s})$	non stationary optimal policy of MDP-DBS_{SL} such that $\pi^{\text{fh}} = \{\pi_t^{\text{fh}}, t = 0, .., SL\}$, predetermined action for state $\mathbf{s}$ observed at day t in π_t^{fh}
$\pi^{\text{IDH}}, \pi^{\text{IDH}}(\mathbf{s})$	stationary policy established by IDH, predetermined action for state $\mathbf{s}$ by π^{IDH}
$C_{\max}$	upper bound on daily cost

Consider π^{fh}. Basically, IDH is designed to determine an action for the observed state in on-line manner. However, using IDH, we can established a policy by solving IDH-GAP for all $\mathbf{s} \in \mathbf{S}$ and then, properly transforming their solutions into the form of the policy.

Letting $V^*(\mathbf{s})$ be the optimal reward-to-go value for state $\mathbf{s} \in \mathbf{S}$, $V^*(\mathbf{s})$ is defined for all $\mathbf{s} \in \mathbf{S}$ as follows:

$$V^*(\mathbf{s}) = C\big(\mathbf{s}, \pi^*(\mathbf{s})\big) + \lambda \sum_{\mathbf{s}_+ \in \mathbf{S}} P\big(\mathbf{s}_+ | \mathbf{s}, \pi^*(\mathbf{s})\big) V^*(\mathbf{s}_+). \tag{3.47}$$

Letting $V^{\mathrm{IDH}}(\mathbf{s})$ be the IDH-reward-to-go value for state $\mathbf{s} \in \mathbf{S}$, $V^{\mathrm{IDH}}(\mathbf{s})$ is defined for all $\mathbf{s} \in \mathbf{S}$ as follows:

$$V^{\mathrm{IDH}}(\mathbf{s}) = C\Big(\mathbf{s}, \pi^{\mathrm{IDH}}(\mathbf{s})\Big) + \lambda \sum_{\mathbf{s}_+ \in \mathbf{S}} P\Big(\mathbf{s}_+ | \mathbf{s}, \pi^{\mathrm{IDH}}(\mathbf{s})\Big) V^{\mathrm{IDH}}(\mathbf{s}_+). \tag{3.48}$$

Letting $\mathcal{V}_t^*(\mathbf{s})$ be the optimal reward-to-go value for state $\mathbf{s} \in \mathbf{S}$ over period from day t to SL, $\mathcal{V}_t^*(\mathbf{s})$ is defined for all $\mathbf{s} \in \mathbf{S}$ as follows:

$$\mathcal{V}_t^*(\mathbf{s}) = C\left(\mathbf{s}, \pi_t^{\mathrm{fh}}(\mathbf{s})\right) + \lambda \sum_{\mathbf{s}_+ \in \mathbf{S}} P\left(\mathbf{s}_+ | \mathbf{s}, \pi_t^{\mathrm{fh}}(\mathbf{s})\right) \mathcal{V}_{t+1}^*(\mathbf{s}_+) \tag{3.49}$$

where

$$\mathcal{V}_{SL}^*(\mathbf{s}) = C\left(\mathbf{s}, \pi_{SL}^{\mathrm{fh}}(\mathbf{s})\right). \tag{3.50}$$

Notice $\mathcal{V}_1^*(\mathbf{s})$ represents the expected future cost over period from day 1 to SL when following the policy π^{fh} in selecting an action for the observed state.

Letting $\mathcal{V}^{\mathrm{IDH}}(\mathbf{s})$ be the IDH-reward-to-go value for state $\mathbf{s} \in \mathbf{S}$ over period from day 1 to SL, $\mathcal{V}^{\mathrm{IDH}}(\mathbf{s})$ is defined as follows:

$$\mathcal{V}^{\mathrm{IDH}}(\mathbf{s}) = \sum_{i=1}^{L} f^l\left(s^l, \pi^{\mathrm{IDH}|l}(s^l)\right) \tag{3.51}$$

where $\pi^{\mathrm{IDH}|l}$ is a collection of decision rules of product lot l for every state which are predetermined in π^{IDH}. It produces product lot l's action given its state s^l. Notice $\mathcal{V}^{\mathrm{IDH}}(\mathbf{s})$ represents the expected future cost over period from day 1 to SL IDH computes.

Compared to the optimal policy of MDP-DBS, IDH's performance is mathematically defined as follows:

$$0 \leq V^{*}(\mathbf{s}) - V^{\mathrm{IDH}}(\mathbf{s}) \leq \frac{C_{max}}{1-\lambda}\lambda^{SL+1} + \frac{2\lambda\epsilon}{1-\lambda} \tag{3.52}$$

where

$$\epsilon \geq \left|\mathcal{V}_1^{*}(\mathbf{s}) - \mathcal{V}^{\mathrm{IDH}}(\mathbf{s})\right| \tag{3.53}$$

For details of proving IDH's performance, refer to Chang *et al*. (2013).

5. Numerical Experiment

In this section, we validate IDH and analyze relocation behaviors of product lots using simulation experiments. Section 5.1 investigates performance of IDH in terms of the optimality gap and the computation time. It compares performance indicators of IDHs based on different ways of estimating future cost: the MP strategy, the BSSPwRuSD strategy, and the BSSPwRuDD strategy. Section 5.2 explores the tuning parameters of IDH, the number of samples and the length of the sample. It shows how IDH's performance changes by different settings of the tuning parameters. Section 5.3 scrutinizes relocation-behaviors of product lots in a dynamic block stacking system and characterizes them based on the inventory level and the row depth.

For numerical experiments, we developed a discrete-event simulation where a block stacking system's operations are represented as a sequence of discrete events. The simulation model assumes at a decision epoch, a series of events occurs in an instant and the system state changes immediately. The series of events refers to sequential operations: observing on the system state $\mathbf{s}$; determining system action $\mathbf{a} \in \mathbf{A_i}$; replenishing and relocating product lots; and then, retrieving unit loads by demand. No event is assumed between two consecutive decision epochs; thus, the simulation time jumps directly from the present decision epoch to the next decision epoch. At a decision epoch under a simulation time frame, the simulation model gives a system

state $\bar{\mathbf{s}}$ and requests a controller determine a system action $\mathbf{a}$. A controller solves IDH-GAP using CPLEX 12.6.3 and returns $\bar{\mathbf{a}}$ to the simulation model. Once $\bar{\mathbf{s}}$ and $\bar{\mathbf{a}}$ are given at a decision epoch, the simulation model randomly defines the system state at the next decision epoch $\bar{\mathbf{s}}_+$ according to the probability distribution of system-state-transition, $P(\mathbf{s}_+|\bar{\mathbf{s}},\bar{\mathbf{a}})$. Each product lot's daily demand follows a Poisson distribution and average daily demand is defined by the instance-generating-procedure introduced in Appendix B of CHAPTER 2.

In the experiments, we used randomly generated instances of three groups, as defined in Table 3.7. For details of the random generation, refer to Appendix B of CHAPTER 2. Each group is distinguished from others in the set of the number of lots and the set of the number of row depth types. Based on the number of variables in IDH-GAP, the instances in Group 1, Group 2, and Group 3 are referred to as small-sized, medium-sized, and large-sized problems. We consider a medium-sized problem to be a practical-sized problem. However, we recognize that large-sized problems exist, but not as widely as medium-sized problems.

Table 3.7: Summary of instances randomly generate for the simulation experiment

		Group 1	Group 2	Group 3
Set of the number of lots		{10, 15, 20}	{30, 40, 50}	{100, 150, 200}
Set of the number of row depth types		{4, 5, 6}	{6, 7, 8}	{8}
Number of instance types		9	9	3
Number of row positions in storage area	Average	14.33	30.88	35.67
	Min	6	38	37
	Max	23	25	34
Number of decision variables in IDH-GAP	Average	75	280	1200
	Min	40	180	800
	Max	120	400	1600

A single instance type is defined by mixing elements of two sets. Taking all combinations of the possible values of the each element, 9, 9, and 3 instance types are defined in Group 1, 2, and 3 respectively. We specify each instance type using the two-tuple ($L|B$) where L and B

respectively indicate the number of lots and the number of storage areas considered. We created one case per a single instance type and replicated five times the simulation experiment of a single case. Thus, a total of 45, 45, and 15 simulation experiments of Group 1, 2 and 3 were performed. For each instance, the number of decision variables of IDH-GAP is computed by $L*B$.

The experiment runs a single simulation model over 260 days of the simulation-time horizon, assuming there are 260 business days in a year. Daily discount factor λ is set as 0.99980274 derived from an annual discount factor of 0.95. All experiments were conducted on an Intel Xeon Processor X5670 (hexa-core, 12M cache, 2.93 GHz) with 24 GB RAM and execution files were run on a UNIX platform.

5.1. Validation of IDH

After simulating, product lots' daily demands and daily inventory levels are known; consequently, we have an instance of Block Stacking Multiple Products with Relocation under Deterministic Demand (BSMPwRuDD), which is deeply investigated in CHAPTER 2. The problem of optimizing BSMPwRuDD is formulated as an integer program and referred to as IP-BSMPwRuDD. For more details of IP-BSMPwRuDD and its solution procedure, refer to CHAPTER 2. Comparing the operating cost of the solution obtained by IDH and by solving IP-BSMPwRuDD, the former is greater than or equal to the latter. This is a natural consequence, because IDH's solution is based on uncertain daily inventory levels and daily demands; whereas, IP-BSMPwRuDD's solution is based on known inventory levels and daily demands over a time horizon. Therefore, in validating IDH, we adopted the objective function value of IP-BSMPwRuDD as a lower bound to the objective function value of IDH's solution.

In this section, we compare the results of solving instances of BSMPwRuSD using IDH based on the MP strategy, the BSSPwRuSD strategy, and the BSSPwRuDD strategy. Hereafter, to

simplify the expression, let *M*, *S*, and *D* indicate the MP strategy, the BSSPwRuSD strategy, and the BSSPwRuDD strategy and be used as a superscript in notations if required. For example, IDH^{M} represents IDH based on the MP strategy.

Let OFV^{i}, $i \in \{M, S, D\}$ be the objective function value of IDH-GAP^{i}, $i \in \{M, S, D\}$. Additionally, OFV^{*} represents the optimal objective function value of IP-BSMPwRuDD and OFV^{LP} indicates the optimal objective function value of the linearly-relaxed IP-BSMPwRuDD. IP-BSMPwRuDD; linearly-relaxed IP-BSMPwRuDD were solved by CPLEX 12.6.3. Notice the following inequality is valid among OFV^{M}, OFV^{S}, OFV^{D}, OFV^{*}, and OFV^{LP}:

$$\text{OFV}^{LP} \leq \text{OFV}^{*} \leq \text{OFV}^{M}, \text{OFV}^{S}, \text{OFV}^{D} \tag{3.54}$$

5.1.1. Optimality gap analysis

In this section, we compare the performance of IDH^{M}, IDH^{S} and IDH^{D} based on the optimality gap and the computation time. For $i \in \{M, S, D\}$, let Gap_{*}^{i} and Gap_{LP}^{i} be the optimality gap computed as follows:

$$\text{Gap}_{*}^{i} = \left(\text{OFV}^{i} - \text{OFV}^{*}\right)/\text{OFV}^{*} \tag{3.55}$$

$$\text{Gap}_{LP}^{i} = \left(\text{OFV}^{i} - \text{OFV}^{LP}\right)/\text{OFV}^{LP} \tag{3.56}$$

For optimality gap analysis, we consider the results of simulation experiments of instances where CPLEX obtains the optimal solution of the corresponding IP-BSMPwRuDD within 24 hours. In the experiment, the following six instances of Group 1 satisfied this criterion: (10|4)-, (15|4)-, (15|5)-, (20|4)-, (10|5)-, and (10|6)-instance. Unlike the first four instances, the last two instances resulted in extremely low OFV^{LP}. Thus, we conducted separate analysis for the first four instances and for the last two instances.

Table 3.8: Comparison of computation times, Gap_*^is, $i \in \{M, S, D\}$, and Gap_{LP}^is, $i \in \{M, S, D\}$ based on experiment results of simulating (10|4)-, (15|4)-, (15|5)-, and (20|4)-instance

		IDH^M	IDH^S	IDH^D
Average computation Time (sec)	Total time	0.13	0.24	28.28
	Sampling time	0.00	0.11	0.11
	Decision making time	0.13	0.13	28.17
Optimality gap based on OFV^* (%) ($\text{Gap}_*^{\cdot}$)	Average	1.43	0.75	0.73
	Max	2.05	1.14	1.14
	Min	0.76	0.42	0.42
Optimality gap based on OFV^{LP} (%) ($\text{Gap}_{LP}^{\cdot}$)	Average	3.73	3.03	3.01
	Max	6.67	6.35	6.29
	Min	1.49	0.69	0.68

Table 3.9: Comparison of computation times, Gap_*^is, $i \in \{M, S, D\}$, and Gap_{LP}^is, $i \in \{M, S, D\}$ based on experiment results of simulating (10|5)- and (10|6)-instance

		IDH^M	IDH^S	IDH^D
Average computation Time (sec)	Total time	0.15	0.38	52.27
	Sampling time	0.00	0.22	0.22
	Decision making time	0.15	0.15	52.04
Optimality gap based on OFV^* (%)	Average	1.21	1.20	1.20
	Max	1.46	1.56	1.56
	Min	1.05	1.00	0.94
Optimality gap based on OFV^{LP} (%)	Average	25.31	25.32	25.31
	Max	43.82	43.99	43.99
	Min	10.46	10.35	10.25

Table 3.8 and Table 3.9 summarizes the experimental results of simulating (10|4)-, (15|4)-, (15|5)-, and (20|4)-instances and simulating (10|5)- and (10|6)-instances. In both tables, the second row contains the computation time of IDH^M, IDH^S and IDH^D required to solve a single instant-action-determination problem in the simulation experiment. The fifth row represents the average of Gap_*^M, Gap_*^S, and Gap_*^D and the eighth row shows the average of Gap_{LP}^M, Gap_{LP}^S, and Gap_{LP}^D.

Based on the small values of average Gap_*^M, Gap_*^S, and Gap_*^D shown in the fifth row of Table 3.8 and Table 3.9, we concluded IDH^M, IDH^S, and IDH^D work well for small-sized instances.

In both tables, IDH^{M} provides the worst solution with the shortest computation time and IDH^{D} gives the best solution with the longest computation time. The computation time of IDH^{D} is not too long for small-sized instances but increases as the number of product lots and the number of storage areas increase. Thus, IDH^{D} may be an impractical solution procedure for large-sized instances. Gap_{*}^{S} is very close to Gap_{*}^{D} and computation time of IDH^{S} is significantly shorter than one of IDH^{D}. Thus, IDH^{S} may be a good alternative to IDH^{D} in solving a large-sized instance.

Comparison of the fifth row and the eighth row in Table 3.8 and in Table 3.9 show OFV* is closer to OFV^{i}s, $i \in \{M, S, D\}$ rather than OFV^{LP}.

Consider the rows concerning Gap_{LP}^{M}, Gap_{LP}^{S}, and Gap_{LP}^{D} in both tables. Interestingly, the values are very large in Table 3.9. This observation can be explained by two keywords, lot splitting and space utilization. Naturally, linearly-relaxed IP-BSMPwRuDD allows lot splitting of a product lot into different storage areas. Therefore, the solution of linearly-relaxed IP-BSMPwRuDD stores no unit load in the extra storage area as long as the inventory level remains less than the capacity of the regular storage area. On the other hand, IDH^{i}s, $i \in \{M, S, D\}$ and IP-BSMPwRuDD assume no lot splitting; thus, their solutions sometimes store unit loads in the extra storage area even though the inventory level is less than the capacity of the regular storage area. In other words, because of the different assumptions concerning lot splitting, the number of unit loads stored in the extra storage area in the solution of linearly-relaxed IP-BSMPwRuDD is resistant to increasing space utilization and the number of unit loads stored in the extra storage area in the solutions of IDH^{i}s, $i \in \{M, S, D\}$ and IP-BSMPwRuDD increases as space utilization increases. Because storing unit loads in the extra storage area incurs penalty cost, for IDH^{i}s, $i \in \{M, S, D\}$ and IP-BSMPwRuDD, increased space utilization results in a solution with larger

penalty costs. Consequently, the greater space utilization, the greater the gap between OFV^{LP} and OFV^{*}, OFV^{M}, OFV^{S}, and OFV^{D}.

For example, Table 3.10 compares the experiment results of simulatinga (10|4)-instance and a (10|5)-instance. In the first row of the table, Gap^{*}_{LP} represents the gap between OFV^{LP} and OFV^{*} computed by $(OFV^{*}\text{-}OFV^{LP})/OFV^{LP}$. Table 3.10 shows, when space utilization is higher, the gaps between OFV^{LP} and OFV^{*}, OFV^{M}, OFV^{S}, and OFV^{D} are larger. In the second row corresponding to the instance of lower space utilization, all indicators of the gap are less than 5% and in the third row corresponding to the instance of higher space utilization, greater than 16%.

Table 3.10: Impact of space utilization on Gap^{*}_{LP} and Gap^{i}_{LP}s, $i \in \{M, S, D\}$ in simulation experiments of (10|4)- and (10|5)-instance

Instance	# of days when space utilization is over 80%	Gap^{*}_{LP} (%)	Gap^{M}_{LP} (%)	Gap^{S}_{LP} (%)	Gap^{D}_{LP} (%)
(10|4)	30.4	2.89	4.51	3.71	3.66
(10|5)	71.2	16.53	18.03	17.90	17.90

5.1.2. Reliability of feasible solution

In this section, we show IDH^{M} and IDH^{S} also work well for practical-sized instances of Group 2 and Group 3 and we compare their performances based on the associated computation time and Gap^{M}_{LP} and Gap^{S}_{LP}. Notice CPLEX cannot obtain an optimal solution of any IP-BSMPwRuDD corresponding to each instance of Group 2 and Group 3 within 24 hours; consequently, OFV^{*} and Gap^{M}_{*} and Gap^{S}_{*} are unavailable. For the analysis, we consider the results of 60 simulation experiments of Group 2 and Group 3 instances.

Table 3.11 summarizes the comparison of Gap^{M}_{LP} and Gap^{S}_{LP}. Considering the observation that OFV^{*} is closer to OFV^{M} and OFV^{S} rather than OFV^{LP}, Gap^{M}_{*} and Gap^{S}_{*} would be less than

the corresponding Gap_{LP}^{M} and Gap_{LP}^{S} given in Table 3.11. Thus, IDH^{M} and IDH^{S} work well for practical-sized instances.

Table 3.11: Comparison of Gap_{LP}^{M} and Gap_{LP}^{S} based on the experiments results of simulating Group 2 instances and Group 3 instances

Instances		Number of simulation experiment	Gap_{LP}^{M}			Gap_{LP}^{S}		
			Average	Max	Min	Average	Max	Min
Group 2	(30|·)	15	5.63	7.12	4.13	5.50	6.94	3.73
	(40|·)	15	4.31	5.28	2.87	4.03	5.01	2.55
	(50|·)	15	4.49	5.26	3.35	4.09	4.89	3.11
Group 3	(100|8)	5	4.79	5.07	4.53	3.04	3.18	2.93
	(150|8)	5	4.92	5.15	4.63	2.27	2.41	2.04
	(200|8)	5	4.43	4.67	4.33	1.43	1.53	1.35

Table 3.12 provides computation times of IDH^{M} and IDH^{S} to solve a single instant-action-determination problem in the simulation experiment. Even though the computation time of IDH^{M} and IDH^{S} increases as the number of product lots increases, it is still very small for practical-sized instances.

Table 3.12: Comparison of computation time of IDH^{M} and IDH^{S} spent to solve a single instant-action-determination problem in experiment of simulating Group 2 instances and Group 3 instances

Instances	Number of simulation experiment	Average time (sec)					
		IDH^{M}			IDH^{S}		
		Total	Sampling	Decision making	Total	Sampling	Decision making
(30|·)	15	0.58	0.00	0.58	1.28	0.66	0.62
(40|·)	15	0.70	0.00	0.70	1.51	0.71	0.81
(50|·)	15	0.84	0.00	0.84	1.71	0.76	0.94
(100|8)	5	0.96	0.00	0.96	1.68	0.85	0.83
(150|8)	5	0.92	0.00	0.92	2.02	0.96	1.06
(200|8)	5	0.44	0.00	0.44	1.56	0.95	0.62

Comparing IDH^{M} and IDH^{S}, IDH^{M} is yields the smallest computation times and IDH^{S} yields the smallest optimality gaps. To solve practical-sized instances of BSMPwRuSD, we recommend using IDH^{S} based on a smaller optimality gap and short computation time even though it is longer than the computation time with IDH^{M}.

5.2. Analysis of tuning parameters of IDH

In this section, we investigate the tuning parameters of IDH, i.e., the number of samples, *SN* and the length of the sample, *SL*. More specifically, a two-way ANOVA analysis was performed to check whether *SN* and *SL* and their interaction have a statistically significant effect on the performance of IDH^{S}. For the results of preliminary analysis regarding the normality assumption of residuals and the homogeneity assumption of variance, refer to Appendix A of this chapter.

A possible hypothesis concerning IDH is a larger number of samples of longer time horizon in solving an instance of BSMPwRuSD results in a better solution by incurring lower operating cost. In other words, IDH with *SN* = 50 and *SL* = 50 may guarantee a better solution compared to IDH with *SN* = 10 and *SL* = 10. In this section, we verify this hypothesis with the results of the ANOVA analysis.

Table 3.13 summarizes the design factors of the two-way ANOVA analysis. We conducted the two-way ANOVA analysis with 9 practical-sized instances of Group 2. The dependent variable is the ratio of Gap_{LP}^{S} to Gap_{LP}^{M}, in order to normalize Gap_{LP}^{S} over different instances of OFV^{LP}. Independent variables are *SN* and *SL*; the set of their levels is {10, 20, 30, 40, 50}. For each instance, we performed experiments for all of each combination of *SN* and *SL*; the experiment with the same combination is replicated three times.

Table 3.13: Design factors of the two-way ANOVA

Design factor		*Specifics*
Instances		(L|B)-instance, L = 30, 40, 50 and B = 6, 7, 8
Dependent variable		$\text{Gap}_{LP}^{S}/\text{Gap}_{LP}^{M}$
Independent variables		*SN* and *SL*
Level of independent variables	*SN*	{10, 20, 30, 40, 50}
	SL	{10, 20, 30, 40, 50}
Experiment plan		Factorial design
Replication		3 times

Table 3.14Table 3.14 summarizes the results of two-way ANOVA carried out on $\text{Gap}_{LP}^{S}/\text{Gap}_{LP}^{M}$ by *SN* level and *SL* level for instances of Group 2. From Table 3.14, for the (30|6) instance, there was a statistically significant effect of *SN* on $\text{Gap}_{LP}^{S}/\text{Gap}_{LP}^{M}$ (p = 0.023) and *SL* on $\text{Gap}_{LP}^{S}/\text{Gap}_{LP}^{M}$ (p = 0.014). The result of Tukey's HSD post hoc test indicates *SL* = 20 was significantly different from *SL* = 30 (p = 0.010); there is no significant difference between any other pairs of *SN* levels. For the (30|7) instance, there was no statistically significant effect of *SN* and *SL* on $\text{Gap}_{LP}^{S}/\text{Gap}_{LP}^{M}$. For the (30|8) instance, there was a statistically significant interaction between the effects of *SN* and *SL* on $\text{Gap}_{LP}^{S}/\text{Gap}_{LP}^{M}$ (p = 0.039). The result of Tukey's HSD post hoc test indicates the pair of *SN*=10 and *SL*=40 and *SN*=20 and *SL*=10 were significantly different from each other (p = 0.045) and, for *SN*=20, *SL*=10 was significantly different from *SL*=40 (p = 0.032).

The results of the two-way ANOVAs for the (30|6)-, (30|7)-, and (30|8)-instances show different *SN* levels and *SL* levels led to statistically different $\text{Gap}_{LP}^{S}/\text{Gap}_{LP}^{M}$s. The difference is, however, observed only in few pairs of *SN* levels and *SL* levels; most pairs of *SN* levels and *SL* levels resulted in statistically the same $\text{Gap}_{LP}^{S}/\text{Gap}_{LP}^{M}$s. Thus, *SN* level and *SL* level determine the performance of IDH^{S}. Figure 3.8 displays boxplots of $\text{Gap}_{LP}^{S}/\text{Gap}_{LP}^{M}$ and *SN* and $\text{Gap}_{LP}^{S}/\text{Gap}_{LP}^{M}$ and *SL* for (30|6)-, (30|7)-, and (30|8)-instances. No clear pattern is revealed in the boxplots.

Table 3.14: Results of two-way ANOVA carried out on the ratio of Gap_{LP}^{S} to Gap_{LP}^{M} by *SN* level and *SL* level

		Df	*Sum Sq*	*Mean Sq*	*F-value*	*Pr* (> *F*)
(30|6)-instance	*SN*	4	.0112	.0028	3.1144	.0230*
	SL	4	.0126	.0031	3.4898	.0137*
	SL:*SL*	16	.0122	.0008	.8495	.0623
	Residuals	50	.0450	.0009		
(30|7)-instance	*SN*	4	.0020	.0005	.4028	.8057
	SL	4	.0007	.0002	.1296	.9709
	SL:*SL*	16	.0143	.0009	.7122	.7684
	Residuals	50	.0629	.0013		
(30|8)-instance	*SN*	4	.0112	.0028	4.6542	.0028*
	SL	4	.0052	.0013	2.1525	.0880
	SL:*SL*	16	.0186	.0012	1.9349	.0388*
	Residuals	50	.0301	.0006		
(40|6) instance	*SN*	4	.0092	.0023	1.6233	.1830
	SL	4	.0187	.0047	3.2934	.0180*
	SL:*SL*	16	.0210	.0013	.9223	.5501
	Residuals	50	.0710	.0014		
(40|7) instance	*SN*	4	.0015	.0004	.3067	.8722
	SL	4	.0063	.0016	1.2743	.2925
	SL:*SL*	16	.0258	.0016	1.3009	.2339
	Residuals	50	.0619	.0012		
(40|8) instance	*SN*	4	.0078	.0019	1.1612	.3391
	SL	4	.0012	.0003	.1776	.9489
	SL:*SL*	16	.0161	.0010	.6032	.8661
	Residuals	50	.0836	.0017		
(50|6) instance	*SN*	4	.0065	.0016	1.0312	.4005
	SL	4	.0127	.0032	2.0208	.1057
	SL:*SL*	16	.0204	.0013	.8138	.6641
	Residuals	50	.0784	.0016		
(50|7) instance	*SN*	4	.0023	.0006	.4480	.7733
	SL	4	.0266	.0066	5.1986	.0014*
	SL:*SL*	16	.0152	.0010	.7434	.7372
	Residuals	50	.0639	.0013		
(50|8) instance	*SN*	4	.0043	.0011	.6996	.5959
	SL	4	.0148	.0037	2.4066	.0617
	SL:*SL*	16	.0146	.0009	.5954	.8723
	Residuals	50	.0768	.0015		

Note: * $p < .05$

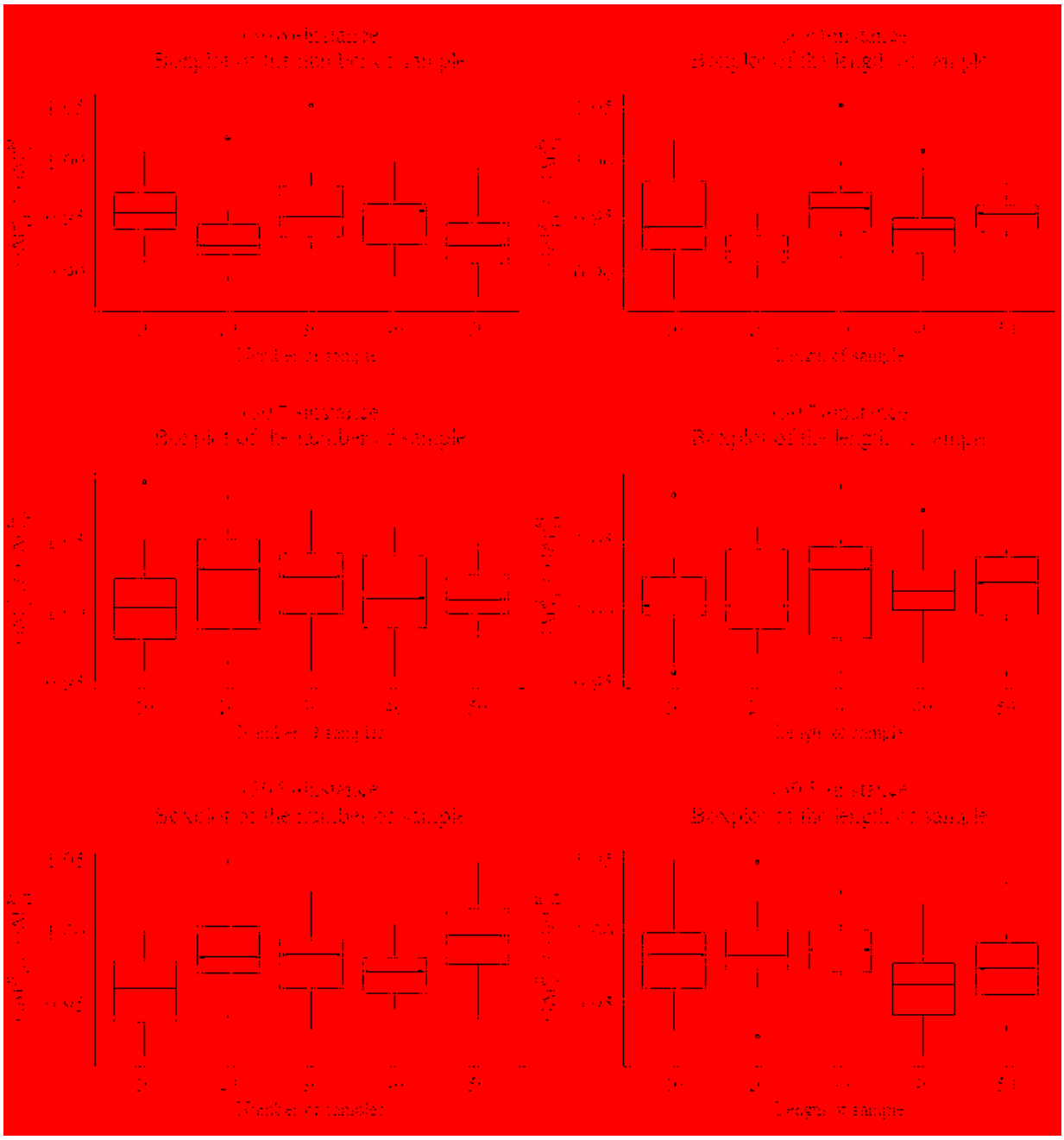

Figure 3.8: Boxplots of the ratio of Gap_{LP}^{S} to Gap_{LP}^{M} and *SN* and the ratio of Gap_{LP}^{S} to Gap_{LP}^{M} and *SL* for (30|6)-, (30|7)-, and (30|8)-instance

Likewise, based on the results of the two-way ANOVAs for (40|6)-, (40|7)-, and (40|8)-instance and (50|6)-, (50|7)-, and (50|8)-instance, we cannot conclude *SN* level and *SL* level controls the performance of IDH^{S}. In addition, all boxplots of $\text{Gap}_{LP}^{S}/\text{Gap}_{LP}^{M}$ and *SN* and

$\text{Gap}_{LP}^{S}/\text{Gap}_{LP}^{M}$ and *SL* for these instances show no clear pattern (the boxplots for (40|6)-, (40|7)-, and (40|8)-instances and (50|6)-, (50|7)-, and (50|8)-instances are given in Appendix A of this chapter).

The results of the two-way ANOVAs indicate in most of cases, different pairs of *SN* level and *SL* level produced statistically the same $\text{Gap}_{LP}^{S}/\text{Gap}_{LP}^{M}$s. This observation was common over all instances regardless of the number of product lots and storage areas considered. It suggests the quality of OFV^{S} is hardly affected by *SN* level and *SL* level and indicates there is no benefit of setting *SN* and *SL* as a larger number. Note the computation time of IDH increases as the value of *SN* and *SL* increases. Based on these findings, we suggest setting *SN* as 10 and *SL* as 10 when using IDH to solve practical-sized instances of BSMPwRuSD.

5.3. Analysis of relocation behavior

In this section, we analyze relocation-behaviors of product lots in a dynamic block stacking system and characterize them based on the inventory level and the row depth. The relocation-behavior indicates whether a product lot is relocated at a decision epoch and, if so, from which deep storage area to which deep storage area. Therefore, in the analysis, the case of changing row depth at replenishment point is not considered. In this section, the analysis is based on data collected from the experiments of simulating (30|6)-, (40|6)-, and (50|6)-instances. Summaries of product lots' relocation-behaviors of every instance are provided in Appendix B of this chapter.

Table 3.15 summarizes the percentage of every scenario of relocation-behavior of a product lot in the experiment of simulating (30|6)-, (40|6)-, and (50|6)-instances. On average, about 75% of product lots remain in their current storage area and about 25% of product lots are relocated at decision epoch. Relocation to a shallower-deep storage area is more frequent than to a deeper-deep storage area.

Table 3.15: Percentage of every scenarios of relocation-behavior of a product lot in the experiment of simulating (30|6)-, (40|6)-, and (50|6)-instance

Scenario of relocation-behavior of a product lot	Percentage (%)
Product lot remains in current storage area	75.05
Product lot relocated into shallower-deep storage area	17.23
Product lot relocated into deeper-deep storage area	5.75
Product lot relocated to extra storage area	1.15
Product lot relocated from extra storage area	0.71
Product lot remains in extra storage area	0.12

Table 3.16 organizes the percentage of cases of relocating a product lot in the experiment of simulating (30|6)-, (40|6)-, and (50|6)-instances. The tenth column of the percentage of relocation from each storage area shows relocation from a deeper storage area is more frequent than from a shallower storage area. For example, the relocation from 20-deep storage area is most frequent. The tenth row of the percentage of relocation to each storage area shows relocation to a very shallow storage area is more frequent than to a less shallow storage area. For example, the relocation to 2-deep storage area is most frequent. The fifth and sixth row show when a current storage area is 5-deep or 10-deep, relocation to a shallower storage area is more frequent than to a less shallow storage area. For example, relocation from a 5-deep storage area to a 2-deep storage area is more frequent than to a 3-deep storage area.

Table 3.16: Percentage of cases of relocating a product lot in the experiment of simulating (30|6)-, (40|6)-, and (50|6)-instance

		Assigned storage area							
		2	3	5	10	15	20	Ex	Sum
Current storage area	2	-	4.28	1.28	0.39	0.31	0.25	1.39	7.9
	3	11.5	-	2.08	0.87	0.53	0.74	0.69	16.41
	5	6.47	4.98	-	0.57	0.94	2.21	0.48	15.65
	10	5.13	4.56	2.04	-	3.89	1.6	0.66	17.88
	15	4.02	3.78	2.03	5.7	-	3.21	0.66	19.4
	20	3.84	3.18	6.24	2.18	3.66	-	0.79	19.89
	Ex	0.89	0.50	0.42	0.29	0.35	0.42	-	2.87
	Sum	31.85	21.28	14.09	10.00	9.68	8.43	4.67	100

Interestingly, among cases of relocation to extra storage area and from extra storage area, the relocation from 2-deep storage area and to 2-deep storage area is most frequent. Considering inventory level of a product lot stored in a 2-deep storage area is low, it implies the IDH makes the assignment of product lots to a storage area where less number of unit loads are relocated from or to extra storage area.

Table 3.17 summarize ratio of average inventory level at cases of relocating to average inventory level at cases of remaining in the experiment of simulating (30|6)-, (40|6)-, and (50|6)-instance Rows corresponding to relocation from each storage area shows when current inventory level is more less than the average inventory level at cases of remaining, a product lot tends to be relocated to more shallower-deep storage area. For example, the ratio corresponding to relocation from 10-deep storage area to 2-deep storage area is less than to 3-deep storage area. In addition, when current inventory level is more greater than the average inventory level at cases of remaining, a product lot is likely to be relocated to more deeper storage area. For example, the ratio corresponding to relocation from 2-deep storage area to 5-deep storage area is greater than to 3-deep storage area

Table 3.17: Ratio of average inventory level at cases of relocating to at cases of remaining in the experiment of simulating (30|6)-, (40|6)-, and (50|6)-instance

		Assigned storage area						
		2	3	5	10	15	20	Ex
Current storage area	2	1	1.03	1.63	2.85	4.03	4.43	0.36
	3	0.29	1	0.67	1.15	1.53	1.91	0.22
	5	0.27	0.43	1	1.18	1.16	1.18	0.34
	10	0.24	0.38	0.55	1	0.93	1.03	0.42
	15	0.24	0.37	0.6	0.67	1	1.01	0.52
	20	0.22	0.3	0.46	0.9	0.83	1	0.51
	Ex	0.51	1.13	1.69	2.61	3.83	4.16	1

Table 3.18: Average inventory level of a product lot when it is relocated to each storage area in the experiment of simulating (30|6)-, (40|6)-, and (50|6)-instance

	Assigned storage area						
Instance	2	3	5	10	15	20	Ex
(30|6)	25.97	37.92	61.82	85.99	103.71	107.77	59.31
(40|6)	16.16	27.35	42.15	71.26	82.87	94.82	24.01
(50|6)	14.46	23.80	37.17	61.31	70.76	75.91	16.66
Ave	17.14	28.12	45.03	70.43	82.77	87.45	27.08

Table 3.18 organizes average inventory level of a product lot when it is relocated to each storage area in the experiment of simulating (30|6)-, (40|6)-, and (50|6)-instances. A product lot with lower (greater) inventory level is relocated to a shallower-deep (deeper-deep) storage area.

The findings of this section provide an insight into the product lot's relocation-behavior. Even though the analysis is only based on data collected from the experiment of simulating (30|6)-, (40|6)-, and (50|6)-instances, most findings are very common over all instances. The existence of patterns in product lots' relocation-behaviors motivates developing a heuristic algorithm for solving practical-sized instances of BSMPwRuSD. For example, based on the observations in Table 3.17, when a product lot is stored in a 10-deep storage area and the ratio of its inventory level to the average inventory level at cases of remaining 10-deep storage area is 0.40, we can make a decision to relocate the product lot to a 3-deep storage area.

6. Conclusions

The first contribution of this research is formulating MDP-DBS, the first optimization model for block stacking multiple products with changeable row depth under stochastic demand. It provides a systematic framework of sequentially establishing a daily DBS plan of determining an assignment of product lots to storage areas for a period over a time horizon when product lots' daily demands are uncertain. In defining a daily DBS plan, by taking account of immediate cost as well as expected future cost, MDP-DBS pursues the global optimum over a planning horizon

rather than the local optimum for a day. Theoretically, MDP-DBS can build an optimal policy by predetermining an optimal action in all states minimizing total expected cost over a planning horizon. In managing a block stacking system operation, an optimal policy of MDP-DBS may work as a guide or propose a good alternative to the controller of the system.

The second contribution of this study is IDH, the solution procedure taking the strategy of the on-line manner by instantly determining an action in the observed single state at a decision epoch. It tackles the computational intractability of solving practical-sized instances of BSMPwRuSD by avoiding searching all reachable future states from the observed single state and enumerating all feasible actions in finding a solution of the problem. In simulation experiments emulating block stacking system operations under stochastic demand, IDH based on the MP manner or the BSSPwRuSD manner, quickly determines a feasible action in the observed system state at every decision epoch. The quality of the solution is guaranteed by the small average and narrow range of the optimality gap.

For future research, finding an optimal value of tuning factors of IDH, length of sample path and the number of the sample path appear to be promising areas for investigation. To achieve the best result in the application of IDH in a real situation, a guide for tailoring IDH is required. Allowing lot splitting in BSMPwRuSD promises to be a very interesting but very challenging research problem; the solution to the problem would establish a more practical DBS plan under stochastic demand setting. The problem of optimizing block stacking multiple products with changeable row depths where some product lots' daily demands are deterministic and demands for other product lots are stochastic is another interesting research problem. We also recommend the problem of aggregating DBS optimization and replenishment scheduling.

7. References

Accorsi, R., Baruffaldi, G., and Manzini, R. (2017) Design and manage deep lane storage system layout. An iterative decision-supporting model. *The International Journal of Advanced Manufacturing Technology*, 92, 57-67.

Ashayeri, J. and Gelders, L. F. (1985). Warehouse design optimization. *European Journal of Operational Research*, 21(3), 285-294.

Bartholdi, J. J., and Hackman S. T. (2014). *Warehouse & Distribution Science*: release 0.96. Georgia Institute of Technology, Atlanta, GA. Retrieved from http://www.warehouse-science.com.

Berry, J. R. (1968). Elements of warehouse layout. *International Journal of Production Research*, 7(2), 105-121.

Boctor, F. F. (2010). Offsetting inventory replenishment cycles to minimize storage space. *European Journal of Operations Research*, 203, 321-325.

Boutilier, C., Dean, T., and Hanks, S. (1999). Decision theoretic planning: structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11, 1-94.

Carlo, H. J., Vis, I. F. A., and Roodbergen, K. J. (2014). Storage yard operations in container terminals: Literature overview, trends, and research directions. *European Journal of Operational Research*, 235(2), 412-430.

Chang, H. S., Hu, J., Fu, M. C., and Marcus, S. I. (2013). *Simulation-Based Algorithms for Markov Decision Processes*, Springer-Verlag, New York, NY

Christofides, N. and Colloff, I. (1972) The rearrangement of items in a warehouse. *Operations Research*, 21, 577-589

De Koster, R. (2010). *Warehouse Math*. L. Kroon, T. L, R. Zuidwijk (eds.), Liber amicorum. In memoriam Jo van Nunen, 179-186, Dinalog Breda.

Derhami, S., Smith, J. S., and Gue, K. R. (2016). A simulation model to evaluate the layout for block stacking warehouse. *Progress in Material Handling Research:2016*, 14th International Material Handling Research Colloquium, http://www.mhi.org/cicmhe/colloquium.

Derhami, S., Smith, J. S., and Gue, K. R. (2017) Optimizing space utilization in block stacking warehouse. *International Journal of Production and Research*, 55(21), 6436-6452.

Fianu, S. and Davis, L. B. (2018). A Markov decision process model for equitable distribution of supplies under uncertainty. European Journal of Operational Research, 264(3), 1101-1115.

Goetschalckx, M., and Ratliff, H. D. (1991). Optimal land depth for single and multiple products in block stacking storage systems. *IIE Transactions*, 23(3), 245-258.

Gong, Y. and De Koster, R. (2011). A review on stochastic models and analysis of warehouse operations. *Logistics Research*, 3(4), 191-205.

Gu, J., Goetschalckx, M., and McGinnis L. F. (2010). Research on warehouse design and performance evaluation: A comprehensive review. *European Journal of Operations Research*, 203(3), 539-549

Hemmi, J. B. (1963). *An Investigation of the Influence of Warehouse Layout on Storage Costs*. Master's thesis. Georgia Institute of Technology, Atlanta, GA.

Jaikumar, R. and Solomon, M. M. (1990) Dynamic operational policies in an automated warehouse. *IIE Transactions*, 22(4), 370-376.

Jang, D., Kim S., and Kim, K. (2013). The optimization of mixed block stacking requiring relocations. *International Journal of Production Economics*, 143(2), 256-262.

Kay, M. G. (2015). *Warehousing*. North Carolina State University, Raleigh, NC. Retrieved from http://www4.ncsu.edu/~kay/Warehousing.pdf.

Kim, K., and Hong, G. (2006). A heuristic rule for relocating blocks. *Computers & Operations Research*, 33, 940-954

Kind, D. A. (1965) Measuring warehouse space utilization. *Transportation and Distribution Management*, 5(7), 23-33.

Kind, D. A. (1975). Elements of space management. *Transportation and Distribution Management*, 15(2), 29-34.

Kooy, E. D. (1981). Making better use of available warehouse space. *Industrial Engineering*, 13(10), 26-30.

Larson, T. N., March, H., and Kusiak, A. (1997) A heuristic approach to warehouse layout with class-based storage. *IIE Transactions*. 29(4), 337-348.

Lee. H., Zhang, S., and White, J. A. (2016). The dynamic block stacking problem with random demand. *Proceedings of the 2016Industrial and System Engineering Research Conference, H. Yang, Z. Kong, and MD Sarder* (eds), Anaheim, CA.

Marsh, W. H. (1979). Elements of block storage design. *International Journal of Production Research*, 17(4), 377-394.

Matson, J. O. (1982). *The Analysis of Selected Unit Load Storage Systems*. Doctoral dissertation. Georgia Institute of Technology, Atlanta, GA.

Matson, J. O., Sonnentag, J. J., White, J. A., and Imhoff, R. C. (2014). An analysis of block stacking with lot splitting. In *Proceedings of the 2014 Industrial and System Engineering Research Conference, Y. Guan and H. Liao* (eds), Montreal, Quebec, Canada, 479-506

Matson, J. O., and White, J. A. (1982). Operational research and material handling. *European Journal of Operational Research*, 11(4), 309-318.

Moder, J. J., and Thornton, H. M. (1965). Quantitative analysis of the factors affecting floor space utilization of palletized storage. *Journal of Industrial Engineering*, 16(1), 8-18.

Moon, I.K., Cha, B.C., and Kim, S.K. (2008). Offsetting inventory cycles using mixed integer programming and genetic algorithm. *International Journal of Industrial Engineering*, 15(3), 245-256.

Muralidharan, B., Linn, R. J., and Pandit, R. (1995) Shuffling heuristics for the storage location assignment in an AS/RS. *International Journal of Production Research*, 33(6), 1661-1672.

Nicol, S. and Chades, I. (2011). Beyond stochastic dynamic programming: a heuristic sampling method for optimizing conservation decisions in very large state spaces. *Methods in Ecology and Evolution*, 2, 221-228

Ouaret, S., Kenne, J., and Gharbi, A. (2018). Production and replacement policies for a deteriorating manufacturing system under random demand and quality. European Journal of Operational Research, 264(2), 623-636.

Peret, L. and Garcia, F. (2004). On-line search for solving Markov decision processes via heuristic sampling. *Proceedings of 16th European Conference on Artificial Intelligence, R. de Mantaras and L. Sattia (eds)*, August 22-27, Valencia, Spain, pp 530-535, IOS press.

Petering, M. E. H., and Hussein, M. I. (2013). A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. *European Journal of Operational Research*, 231(1), 120-130.

Powell, W. (2011). *Approximate Dynamic Programming: Solving the Curse of Dimensionality*. 2nd edition, Wiley & Sons, Inc., Hoboken, NJ.

Puterman, M. L. (2005). *Markov Decision Process: Discrete Stochastic Dynamic Programming*, Wiley & Sons, Inc., Hoboken, NJ

Roberts, S. D. (1968). *Warehouse Size and Design*. Doctoral dissertation. Purdue University, West Lafayette, IN.

Ross, P (1993). Basics of pallet storage systems: Part 1. *Material Handling Engineering*, 48(4), 68-69.

Rust, J. (1997). Using randomization to break the curse of dimensionality. *Econometrica*, 65(3), 487-516.

Sadiq, M., Landers, T. L., and Taylor, G. D. (1996) An assignment algorithm for dynamic picking systems. *IIE Transactions*, 28, 607-616

Sonnentag, J. J., Imhoff, R. C., White, J. A., and Matson J. O. (2014) A consideration of the block stacking multi-product storage problem. In *Proceedings of the 2014 Industrial and System Engineering Research Conference, Y. Guan and H. Liao* (eds), Montreal, Quebec, Canada, 3221-3230

Thornton, H. M. (1961). *Factors Affecting Space Efficiency of Palletized Storage*. Master's thesis. Georgia Institute of Technology, Atlanta, GA.

Tompkins, J. A., White, J. A., Bozer, Y. A., and Tanchoco, J. M. A. (2010) *Facility Planning,* 4th edition. John Wiley & Sons, Inc., New York, NY.

White, D. J. (1993). A survey of application of Markov decision process. *The Journal of the Operational Research Society*, 44(11), 1073-1096.

White, J. A., Sonnentag, J. J., and Matson J. O. (2013). New insights regarding block stacking. In *Proceedings of the 2013 Industrial and System Engineering Research Conference, A Krishnamurthy and W.K.V. Chan* (eds), San Juan, Puerto Rico, 1225-1234.

Yang, J. and Kim, K. (2006). A grouped storage method for minimizing relocation in block stacking systems. *Journal of Intelligent Manufacturing*, 17(4), 453-463.

8. Appendices

8.1. Supplement of the ANOVA analysis in Section 5.2

Table 3.19 summarizes the results of Levene's test. Because the *p* value is greater than 0.05 for all instances, the homogeneity assumption of variance is satisfied for all instances.

Table 3.19: Results of Levene's test for each instance of Group 2

Instance	*Df*	*F-value*	*Pr* (> *F*)
(30|6)-instance	24	.5903	.9189
(30|7)-instance	24	.5532	.9415
(30|8)-instance	24	.2375	.9998
(40|6)-instance	24	.4443	.9833
(40|7)-instance	24	.3783	.9942
(40|8)-instance	24	.5668	.9337
(50|6)-instance	24	.7055	.8221
(50|7)-instance	24	.5673	.9334
(50|8)-instance	24	.6666	.8595

Figure 3.9 shows the normality plot of the residuals for each instance, in which the quantiles of the residuals are plotted against the quantiles of the normal distribution. In all graphs of Figure 3.9, all the points fall approximately along the straight reference line and, thus, the assumption of normal distribution of residuals is satisfied for each instance.

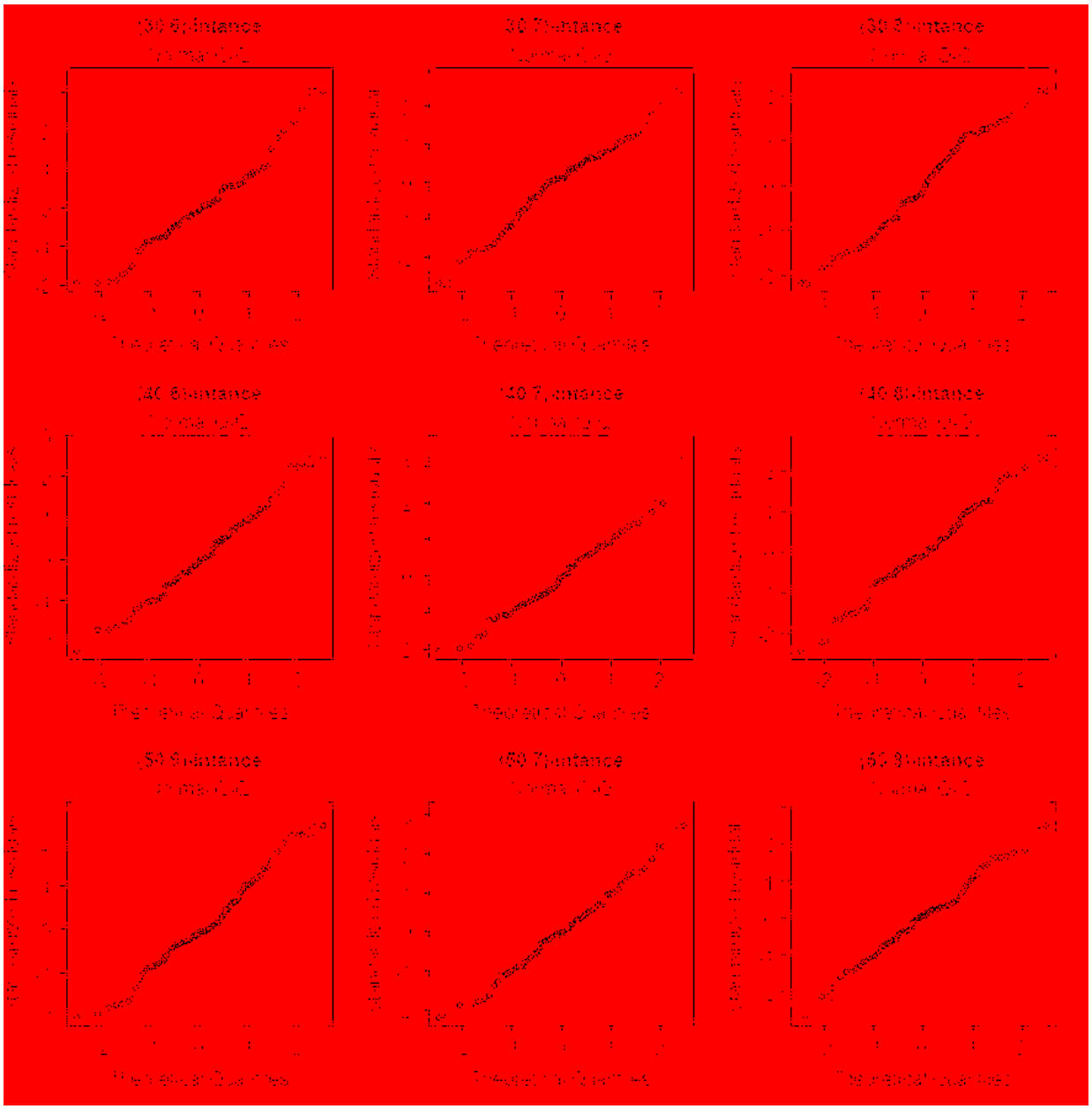

Figure 3.9: Normal Q-Q graph of each instance of Group 2

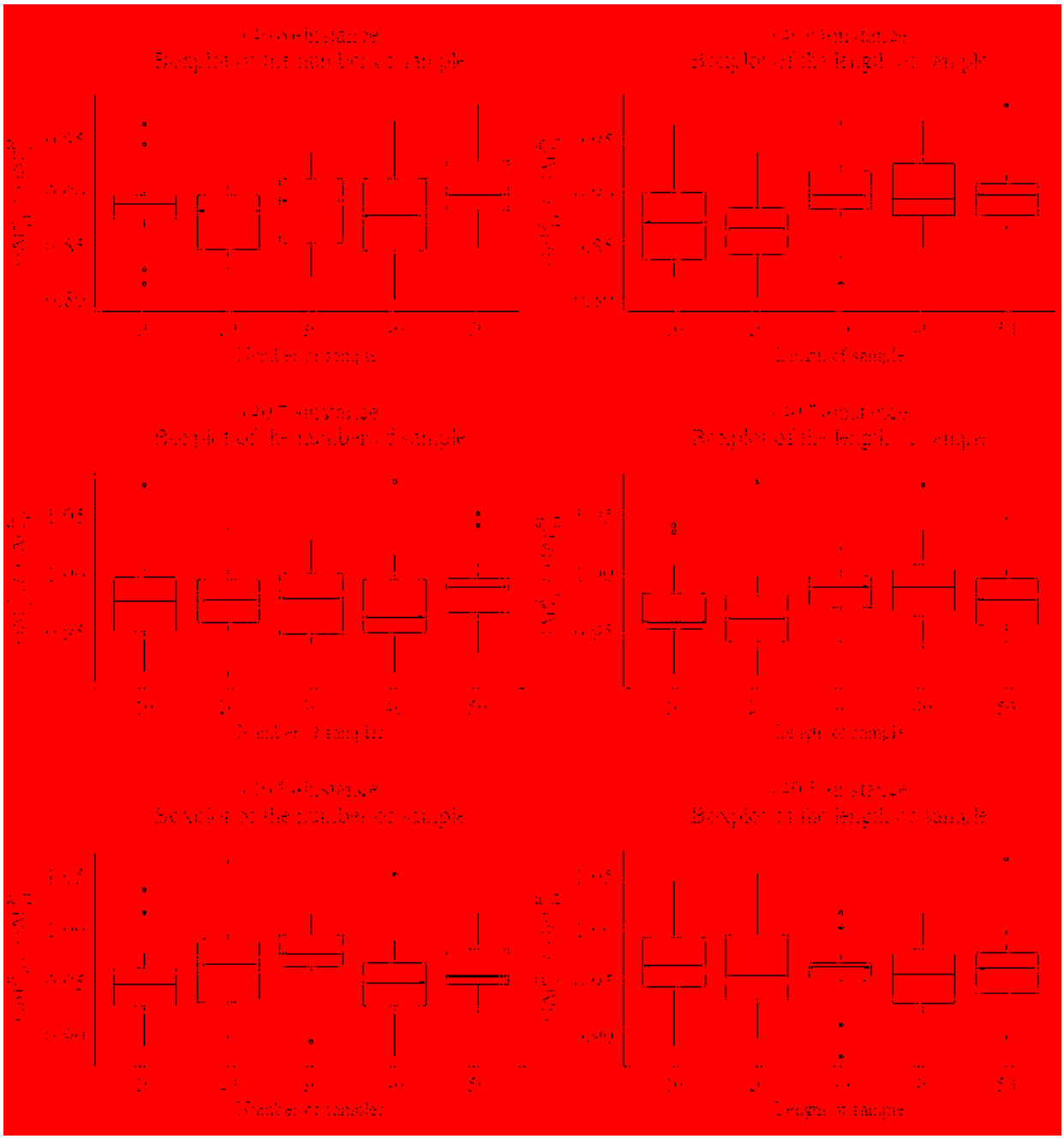

Figure 3.10: Boxplots of the ratio of Gap_{LP}^{S} to Gap_{LP}^{M} and *SN* and the ratio of Gap_{LP}^{S} to Gap_{LP}^{M} and *SL* for (40|6)-, (40|7)-, and (40|8)-instance

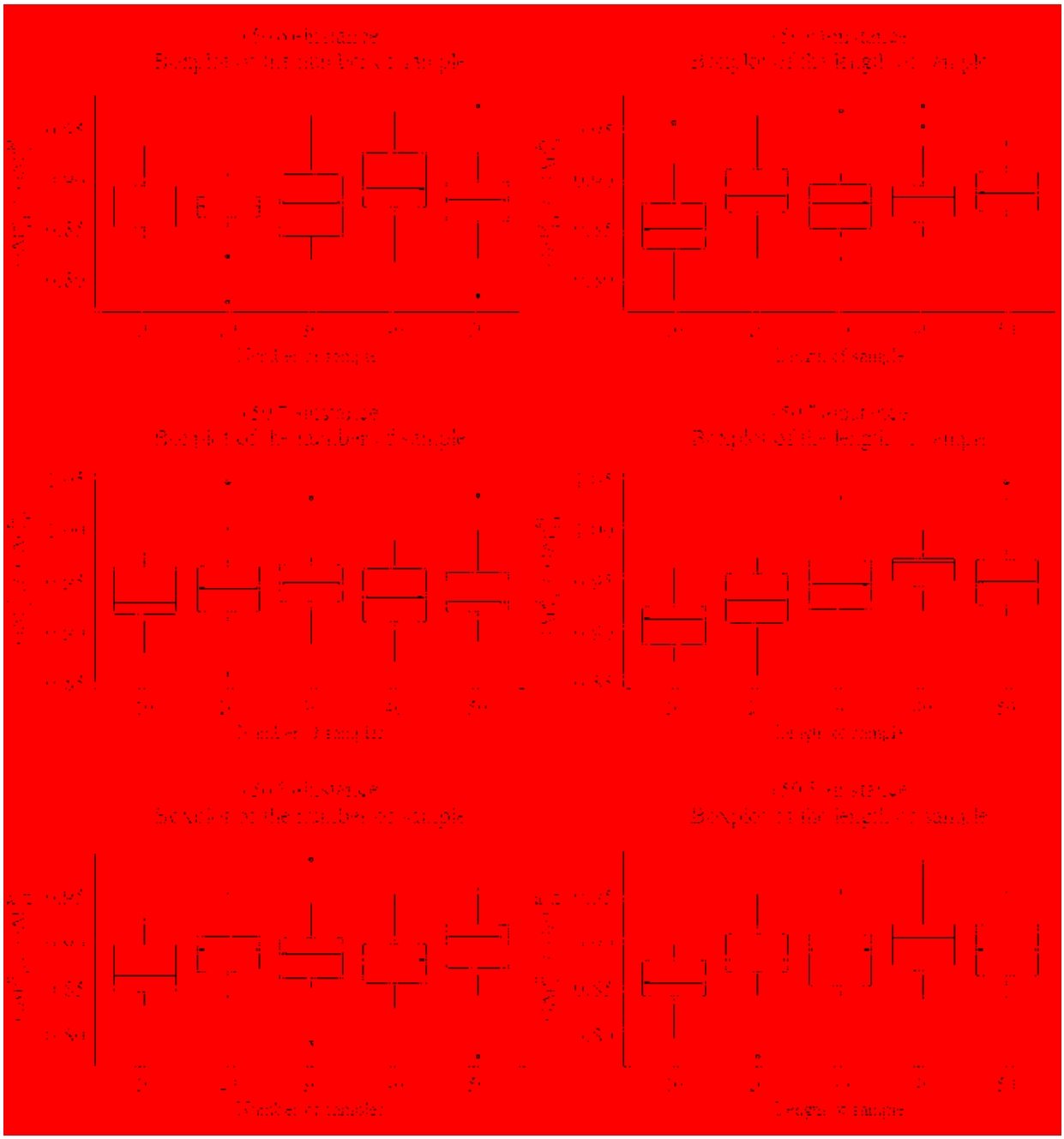

Figure 3.11: Boxplots of the ratio of Gap_{LP}^{S} to Gap_{LP}^{M} and *SN* and the ratio of Gap_{LP}^{S} to Gap_{LP}^{M} and *SL* for (50|6)-, (50|7)-, and (50|8)-instance

8.2. Summary of product lots' behaviors in dynamic block stacking system

Table 3.20: Number of every case of assigning a storage area to a product lot given its current storage area in the experiments of simulating (10|4)-, (15|4)-, and (20|4)-instance

		Assigned storage area					
		2	3	5	10	Ex	Sum
Current storage area	2	11403	375	103	17	44	11942
	3	3209	8682	756	66	9	12722
	5	1314	2828	11054	119	11	15326
	10	608	899	1176	9416	6	12105
	Ex	26	13	8	4	3	54
	Sum	16560	12797	13097	9622	73	52149

Table 3.21: Average of inventory levels at every case of assigning a storage area to a product lot given its current storage area in the experiments of simulating (10|4)-, (15|4)-, and (20|4)-instance

		Assigned storage area					
		2	3	5	10	Ex	Ave
Current storage area	2	3.60	7.54	12.80	24.06	3.02	3.83
	3	4.86	10.81	14.46	25.39	9.22	9.60
	5	5.62	9.70	15.86	23.10	13.55	13.90
	10	8.91	12.80	13.24	23.58	24.67	21.04
	Ex	4.19	8.77	19.88	20.00	4.33	8.80
	Ave	4.20	10.61	15.52	23.58	7.21	12.20

Table 3.22: Number of every case of assigning a storage area to a product lot given its current storage area in the experiments of simulating (10|5)-, (15|5)-, and (20|5)-instance

		Assigned storage area						
		2	3	5	10	15	Ex	Sum
Current storage area	2	2813	389	268	253	74	102	3899
	3	1275	5074	843	211	135	129	7667
	5	1016	1495	7805	288	249	95	10948
	10	823	862	653	11103	525	114	14080
	15	457	776	840	736	12821	128	15758
	Ex	108	98	122	90	96	83	597
	Sum	6492	8694	10531	12681	13900	651	52949

Table 3.23: Average of inventory levels at every case of assigning a storage area to a product lot given its current storage area in the experiments of simulating (10|5)-, (15|5)-, and (20|5)-instance

		Assigned storage area						
		2	3	5	10	15	Ex	Ave
Current storage area	2	15.63	15.05	24.64	41.52	39.70	10.34	18.19
	3	9.63	34.74	25.73	46.64	55.29	17.44	29.97
	5	16.03	29.84	59.93	65.42	71.93	30.20	51.91
	10	28.24	35.76	51.58	84.02	93.62	50.70	76.39
	15	22.13	40.30	57.11	83.35	106.71	67.26	96.93
	Ex	13.69	24.02	43.31	55.96	69.40	42.45	40.77
	Ave	16.54	33.49	55.36	81.89	104.48	37.00	66.03

Table 3.24: Number of every case of assigning a storage area to a product lot given its current storage area in the experiments of simulating (10|6)-, (15|6)-, and (20|6)-instance

		Assigned storage area							
		2	3	5	10	15	20	Ex	Sum
Current storage area	2	1496	290	146	73	49	34	52	2140
	3	698	3415	210	145	90	74	45	4677
	5	509	602	5583	188	177	264	94	7417
	10	449	663	418	8146	659	256	114	10705
	15	354	553	383	1005	9945	410	134	12784
	20	279	467	1125	431	481	11398	225	14406
	Ex	52	91	102	103	145	177	132	802
	Sum	3837	6081	7967	10091	11546	12613	796	52931

Table 3.25: Average of inventory levels at every case of assigning a storage area to a product lot given its current storage area in the experiments of simulating (10|6)-, (15|6)-, and (20|6)-instance

		Assigned storage area							
		2	3	5	10	15	20	Ex	Ave
Current storage area	2	37.96	40.99	68.60	76.52	76.96	87.26	38.54	43.46
	3	40.50	71.95	69.12	103.92	125.80	140.00	61.07	70.13
	5	42.91	84.85	122.53	133.58	144.51	170.22	95.63	116.17
	10	54.93	95.64	137.90	206.01	213.97	270.39	149.65	191.60
	15	58.10	102.96	159.28	189.01	257.11	293.53	173.10	236.93
	20	64.90	119.22	134.54	290.82	273.56	287.71	175.84	263.85
	Ex	35.90	66.81	114.04	139.75	171.94	198.29	178.94	146.66
	Ave	44.85	80.71	124.29	203.51	250.75	282.43	147.21	194.24

Table 3.26: Number of every case of assigning a storage area to a product lot given its current storage area in the experiments of simulating (30|6)-, (40|6)-, and (50|6)-instance

		Assigned storage area							
		2	3	5	10	15	20	Ex	Sum
Current storage area	2	9978	1484	442	134	108	86	481	12713
	3	3985	11447	720	303	183	258	239	17135
	5	2243	1727	14445	196	324	767	167	19869
	10	1778	1581	706	19786	1346	553	228	25978
	15	1393	1310	705	1975	23507	1111	230	30231
	20	1332	1101	2162	754	1267	25540	275	32431
	Ex	307	172	147	99	121	145	163	1154
	Sum	21016	18822	19327	23247	26856	28460	1783	139511

Table 3.27: Average of inventory levels at every case of assigning a storage area to a product lot given its current storage area in the experiments of simulating (30|6)-, (40|6)-, and (50|6)-instance

		Assigned storage area							
		2	3	5	10	15	20	Ex	Ave
Current storage area	2	12.95	13.32	21.14	36.91	52.19	57.43	4.72	13.86
	3	11.69	40.57	27.23	46.80	61.95	77.34	9.07	33.74
	5	16.35	26.06	61.01	72.28	71.07	71.73	20.46	53.28
	10	21.68	34.49	50.44	91.54	85.48	94.05	38.86	81.44
	15	23.93	37.56	60.28	67.30	100.77	101.97	52.12	91.04
	20	23.64	32.34	50.24	97.07	90.25	108.24	55.15	96.91
	Ex	8.53	18.97	28.40	43.90	64.50	69.95	16.82	30.41
	Ave	15.15	35.69	56.98	88.39	98.53	106.10	26.14	70.67

Table 3.28: Number of every case of assigning a storage area to a product lot given its current storage area in the experiments of simulating (30|7)-, (40|7)-, and (50|7)-instance

		Assigned storage area								
		2	3	5	10	12	15	20	Ex	Sum
Current storage area	2	5814	1024	361	169	178	150	49	314	8059
	3	2463	9582	550	242	232	201	149	224	13643
	5	1587	1229	13884	186	396	540	237	217	18276
	10	1207	1656	756	18060	1375	478	406	133	24071
	12	1085	1434	741	1879	17746	579	816	215	24495
	15	571	677	1983	403	775	19432	745	249	24835
	20	674	836	524	828	939	742	22071	288	26902
	Ex	240	148	117	112	171	203	171	226	1388
	Sum	13641	16586	18916	21879	21812	22325	24644	1866	141669

Table 3.29: Average of inventory levels at every case of assigning a storage area to a product lot given its current storage area in the experiments of simulating (30|7)-, (40|7)-, and (50|7)-instance

		Assigned storage area								
		2	3	5	10	12	15	20	Ex	Ave
Current storage area	2	20.54	21.88	36.33	58.08	59.22	57.55	67.96	7.70	23.54
	3	15.56	43.69	36.27	66.50	73.63	79.00	75.45	21.36	39.73
	5	22.02	33.09	67.81	81.65	81.51	99.14	99.79	39.28	62.94
	10	39.51	54.33	78.57	128.07	110.41	132.38	137.64	64.14	115.89
	12	35.71	57.18	83.11	104.63	152.29	142.65	149.80	64.22	134.73
	15	37.40	54.04	79.11	130.97	126.15	155.75	184.95	93.08	143.05
	20	37.31	52.15	83.38	128.02	136.85	150.11	180.00	97.11	165.75
	Ex	16.54	35.90	50.32	65.55	79.36	93.98	99.35	38.62	58.27
	Ave	24.16	44.57	68.83	124.17	144.61	151.44	176.27	52.48	113.39

Table 3.30: Number of every case of assigning a storage area to a product lot given its current storage area in the experiments of simulating (30|8)-, (40|8)-, and (50|8)-instance

		Assigned storage area									
		2	3	5	8	10	12	15	20	Ex	Sum
Current storage area	2	4684	474	295	540	104	146	107	93	211	6654
	3	1273	7529	972	349	154	193	130	117	237	10954
	5	1050	2094	12773	523	175	288	214	144	183	17444
	8	1788	1028	1165	14410	392	294	378	282	140	19877
	10	742	742	564	672	16035	337	278	607	213	20190
	12	503	773	656	469	298	17494	1202	356	264	22015
	15	361	610	626	372	209	1577	17768	366	240	22129
	20	336	403	337	550	1506	388	405	19284	265	23474
	Ex	175	168	139	153	149	180	185	212	294	1655
	Sum	10912	13821	17527	18038	19022	20897	20667	21461	2047	144392

Table 3.31: Average of inventory levels at every case of assigning a storage area to a product lot given its current storage area in the experiments of simulating (30|8)-, (40|8)-, and (50|8)-instance

		Assigned storage area									
		2	3	5	8	10	12	15	20	Ex	Ave
Current storage area	2	39.77	35.14	59.04	75.61	79.39	78.66	77.05	84.11	19.30	45.25
	3	19.15	52.23	52.59	66.48	82.90	83.35	82.70	95.07	29.85	50.19
	5	26.65	53.46	104.05	90.05	98.05	107.50	114.63	131.86	47.83	92.66
	8	64.48	70.81	97.61	172.45	143.42	159.79	155.17	186.44	67.27	151.46
	10	48.42	62.93	90.09	134.14	175.65	176.51	175.97	213.18	101.40	163.42
	12	41.74	65.45	89.42	112.78	146.51	185.05	186.93	212.66	111.76	172.33
	15	44.82	66.86	94.42	125.61	184.59	185.52	230.09	244.99	132.07	212.55
	20	52.00	78.28	119.51	152.26	191.68	224.12	263.16	283.98	169.05	263.11
	Ex	26.78	51.24	66.39	91.28	113.72	124.59	133.76	181.70	55.50	93.92
	Ave	41.17	55.92	98.66	159.86	173.42	182.04	222.36	274.92	84.63	163.49

Table 3.32: Number of every case of assigning a storage area to a product lot given its current storage area in the experiments of simulating (100|8)-, (150|8)-, and (200|8)-instance

		Assigned storage area									
		2	3	5	8	10	12	15	20	Ex	Sum
Current storage area	2	50134	3439	1013	288	59	110	51	77	2592	57763
	3	17195	37101	6148	1308	120	162	160	367	951	63512
	5	6809	13449	46452	1462	169	1243	206	337	460	70587
	8	4635	5099	6435	48750	996	978	1377	1388	329	69987
	10	1689	1836	2285	4781	46264	1054	739	690	152	59490
	12	651	2148	1305	4035	3296	59257	1882	3207	195	75976
	15	432	772	962	812	805	8140	57638	1336	175	71072
	20	475	402	619	1616	2891	1383	3462	65132	127	76107
	Ex	1015	343	207	175	85	125	75	68	1501	3594
	Sum	83035	64589	65426	63227	54685	72452	65590	72602	6482	548088

Table 3.33: Average of inventory levels at every case of assigning a storage area to a product lot given its current storage area in the experiments of simulating (100|8)-, (150|8)-, and (200|8)-instance

		Assigned storage area									
		2	3	5	8	10	12	15	20	Ex	Ave
Current storage area	2	5.55	9.21	19.02	48.54	45.20	53.01	41.04	63.92	2.27	6.31
	3	5.22	15.90	16.84	27.87	50.97	48.38	45.33	57.38	4.62	13.64
	5	8.08	12.52	26.58	37.90	54.35	49.26	55.14	57.21	7.94	22.92
	8	20.55	19.75	18.80	41.43	62.13	43.93	54.52	63.55	16.64	37.30
	10	15.62	18.79	26.83	29.03	54.86	67.69	79.14	72.30	22.44	50.13
	12	17.69	26.72	27.67	28.73	29.98	52.41	68.07	55.89	28.75	49.20
	15	20.38	23.84	45.40	32.56	42.77	43.64	67.82	96.51	32.91	63.75
	20	24.99	33.35	53.38	50.13	66.04	55.84	42.75	78.27	44.58	74.35
	Ex	3.46	10.39	15.01	21.71	36.04	30.26	42.55	48.81	4.61	9.53
	Ave	6.99	15.76	25.30	39.41	53.86	51.50	66.21	77.03	7.21	40.88

CHAPTER 4. Conclusions and Future Research

In this research, we investigate block-stacking operations. Specifically, optimization models are developed for block-stacking operations when the relocation of unit loads is allowed. Assuming changeable row depth instead of permanent row depth, this paper is distinguished from conventional block stacking studies. Optimization models are mathematically formulated and solution procedures are developed to solve practical-sized instances of block-stacking operations.

1. Conclusions

In CHAPTER 2, under the assumption of deterministic demand, the optimization problem of finding the minimum-cost DBS plan was formulated. The problem was modeled using integer programming as a variation of the unsplittable multi-commodity flow problem. The solution of the problem built a DBS plan of defining the assignment of product lots to storage areas each day over a planning horizon. Compared to SBS, we found DBS requires less storage capacity and incurs less operating cost. DBS uses floor space efficiently by timely relocation of product lots. It not only alleviates honeycomb loss and enables the product lot to yield occupied storage locations to another product lot if required. The merit of DBS is magnified when storage capacity is relatively insufficient based on the inventory level. For practical-sized instances, DH solves their corresponding optimization problems in a reasonable time and guarantees a feasible DBS plan. The small average and narrow range of the optimality gaps vouches for the quality of the solution.

In CHAPTER 3, under the assumption of stochastic demand, the optimization problem of determining the minimum-cost DBS plan was formulated. The problem was formulated as a discrete time, finite horizon, discrete event MDP model. MDP-DBS provides a systematic

framework of sequentially establishing a daily DBS plan of defining an assignment of product lots to storage area for a period under the setting of uncertain daily demand. By taking into account immediate cost as well as expected future cost in determining a daily DBS plan; MDP-DBS pursues the global optimum over a planning horizon rather than the local optimum for a day. In simulation experiments emulating block stacking system operations under stochastic demand, IDH based on the MP manner or the BSSPwRuSD manner quickly determines a feasible action in the observed system state at every decision epoch. The quality of the solution is guaranteed by the small average and narrow range of the optimality gap.

2. Practical Application of the Research

Generally, in block stacking system operations, relocating unit loads is rarely considered. Its drawback such as additional material handling and difficulty in inventory tracking easily stands out whereas its benefit like storage space saving and reduced honeycombing is inconspicuous. The findings of CHAPTER 2 show the timely relocation of the proper number of unit loads returns considerable benefit in block stacking system operations.

Based on our research, we recommend adopting a proper operational policy according to the storage capacity and the inventory level of a block stacking system. When the storage capacity is relatively sufficient based on the inventory level, we suggest taking SBS as the operational policy. Compared to DBS, SBS has no significant demerits in this case and is easier to implement. When storage capacity is relatively insufficient, considering the inventory level, we suggest taking DBS as the operational policy. Compared to SBS, even though DBS is harder to implement, DBS can use storage space more efficiently and save operating cost. If the manager is reluctant to adopt DBS, we suggest use SDBS instead. As a kind of hybrid policy of DBS and SBS, compared to SBS, SDBS uses storage space more efficiently but is more difficult to

implement; compared to DBS, SDBS is easier to implement but uses storage space less efficiently.

In our research, operating cost is computed based on the expected row position of the unit loads. In practice, present row position and assigned row position of each unit load may be known and thus, operating cost have to be computed based on not expected row position but exact row position of the unit load.

3. Future Research

In developing BSMPwRuDD and BSMPwRuSD, we assume no lot splitting. Therefore, consideration of lot splitting in BSMPwRuDD and BSMPwRuSD would be welcome as a topic of future research. Another interesting research topic is optimizing block stacking multiple products with changeable row depth where some product lots' daily demand is deterministic and the others are stochastic. Aggregating the problems of DBS optimization and replenishment scheduling and integrating the problems of DBS optimization and block stacking facility design are other interesting issues for future research.

APPENDIX A. Daily Operating Cost Model of Block Stacking

1. Operating Cost of Block Stacking

Block stacking is a storage method used for unitized products. Unlike rack storage, block stacking consists of stacking loads on top of each other. Stacks of unit loads are arranged back-to-back in a row and the rows of the stacks are positioned next to each other in a storage area. Figure A.1 illustrates how stacks and rows are aligned. See Tompkins et al. (2010) for a brief description of the block stacking storage method.

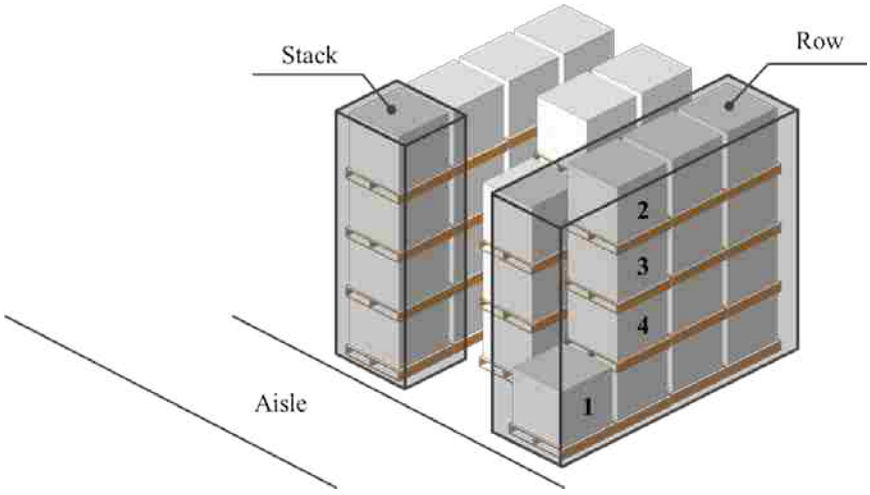

Figure A.1: Arrangement of stacks and rows

The objective of most block-stacking studies is to determine the arrangement of stacks and rows that minimizes the cost of space and/or material handling. Space cost is based on a given layout and/or the average or total dedicated space for storing lots over a time horizon. Material handling cost is based on distance traveled with a given layout or arrangement of unit loads.

An objective function based only on space is more popular than one including space and material handling. No studies employed an objective function based solely on material handling. Table A. categorizes existing papers according to the type of objective function characterized by

cost elements considered. The entries of the "objective function" column indicate which cost elements are included in the objective function:

- FM floor space and material handling
- FN only floor space
- VM volume space and material handling
- VN only volume space

Table A.1: Categorized papers of the block stacking according the feature of the objective function

Objective function	Papers		
FN	Thorton (1961)	Hemmi (1963)	Kind (1965, 1975)
	Moder et al. (1965)	Roberts (1968)	Kooy (1981)
	Matson (1982)	Rickles et al. (1985)	Goetschalckx et al. (1991)
	Koster (2010)	Bartholdi et al. (2014)	Kay (2015)
FM	Roberts (1968)	Matson (1982)	Larson et al. (1997)
	White et al. (2013)	Matson et al. (2014)	Sonnentag et al. (2014)
VN	Roberts (1968)	Kay (2015)	Derhami et al. (2017)
VM	Berry (1968)		

This study develops a daily operating cost model for a single product lot and includes both space and material handling cost. It differs from existing objective function models because it includes the cost of relocating unit loads. Given information such as storage location and inventory level, the daily operating cost of a single product lot is computed based on the following daily operations: storage, replenishment, retrieval, and relocation. The model provides daily costs of the product lot's operating alternatives and is suitable for day-to-day decision making. The daily operating cost of block stacking with multiple products can be computed by summing each product lot's daily operating cost over the set of product lots.

The following sections are organized as follows. Section 2 introduces a typical layout of a block-stacking storage system. Section 3 describes operations of block stacking. In Section 4, the cost model is developed based on the delineated layout and operations in Section 2 and 3. Section 5 evaluates the developed cost model in Section 4.

2. Typical Layout of a Block-Stacking System

A block stacking storage system consists of storage areas having the same or different depths and lengths. Figure A.2 illustrates a typical layout of a block stacking storage system. It consists of six storage areas of the same length and different depths. When the length and depth are measured by the number of unit loads, the storage areas can be referred to as 15-length and 10-deep, 8-deep, 2-deep, 3-deep, 5-deep, and 12-deep storage areas, respectively. The depth and length of the storage area determine the maximum number of stacks in a row in the storage area and the maximum number of rows in the storage area, respectively. For the example, in the 12-deep and 15-length storage area of Figure A.2, 12 stacks are positioned in a row and a maximum of 15 rows are accommodated in the storage area.

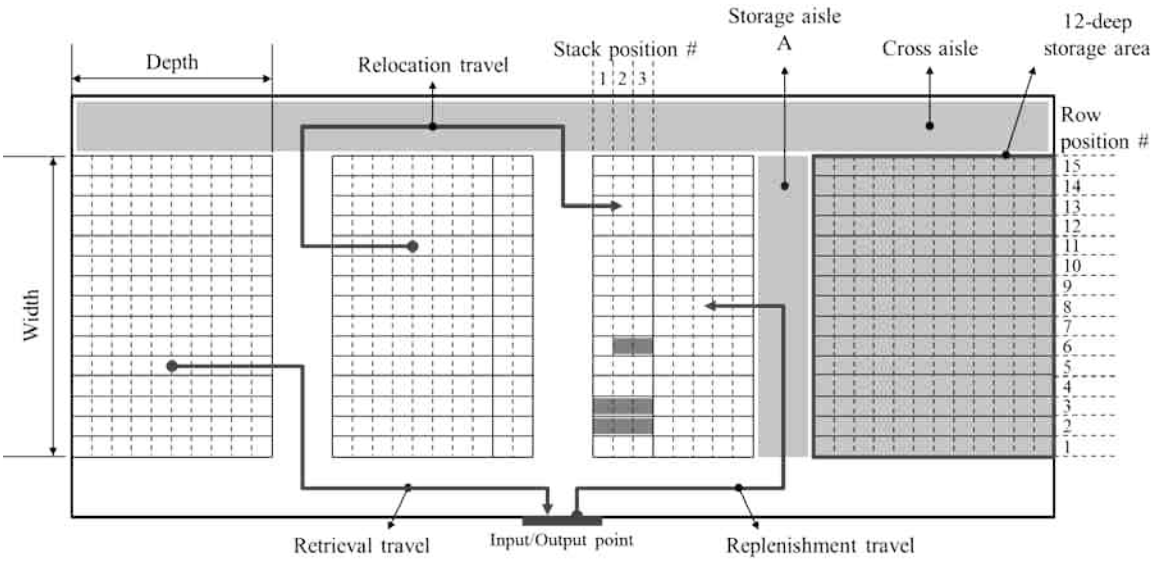

Figure A.2: A typical layout of a block-stacking storage area

In Figure A.2, a single input/output point is located at the center of the lower boundary. Two cross aisles run along the upper and the lower side of the system and storage aisles are perpendicularly connected to the cross aisles. Two-way travel is supposed in these aisles. We assume each storage aisle is designated to serve storage areas on either side of the storage aisle; access to a storage area is only possible from the designated storage aisle. For example, in Figure A.2, storage aisle A serves 5-deep and 12-deep storage area.

In the block stacking storage system, it is explicitly designated where a stack and a row are located. We refer to the location reserved for storing the stack and the row as the *stack position* and the *row position*. We define a *full stack* as a stack fully occupying the volume of space reserved for the stack position and a *partial stack* as a stack partially occupying the space. Similarly, a *full row* and a *partial row* are defined. Stack positions in a row are numbered in increasing order from the closest position to the storage aisle reserved for the row; row positions in a storage area are numbered in increasing order from the closest position to the input/output point. Figure A.2 illustrates the numbering of stack positions of the 3-deep row and row positions of 12-deep storage areas.

3. Assumed Block-Stacking Operations

In developing a daily operating cost model of a single product lot, we consider four daily operations: storage, replenishment, retrieval, and relocation. When the day is divided into business hours and non-business hours, retrieval occurs during business hours; replenishment and relocation occur during non-business hours.

Unit loads of the product lot are stored in an assigned storage area during a day, occupying space and consuming storage capacity. A product lot's storage area for a day is determined at the end of business hours the previous day. A random storage policy is assumed in assigning row

positions to the product lot. Lot splitting, by placing unit loads of single lot in different storage areas, is not allowed. To avoid unnecessarily moving unit loads during retrieval, unit loads of different products and even unit loads of different lots of the same product are not stored in the same row. Consider the row with numbered unit loads in Figure A.1; if unit loads of different lots are consolidated in the row and unit load 4 is to be retrieved, a worker must remove unit loads 1, 2, and 3 blocking unit load 4 before retrieving unit load 4. Then, unit loads 1, 2, and 3 must be returned to the row. However, by prohibiting consolidating unit loads of different lots in a row, unusable space exists in the partial row until it is completely emptied; the space loss is called honeycombing loss. To summarize, this operational rule achieves efficient operations at the cost of honeycombing loss.

A replenishment order for a product is placed at the end of business hours if its inventory level reaches zero. Instantaneous replenishment is assumed, resulting in unit loads of the product arriving at the input point and placement in the storage area assigned. In Figure A.2, replenishment travel from the input point to 5-deep storage area is illustrated. Unit loads are located from the deepest stack position in a row and from the row position closest to the input/output point among the assigned row positions. Assuming a product lot is stored in 5-deep storage area and row positions 2 and 8 are assigned to the lot, unit loads fill from stack position 5 in row position 2. The stack position and row position are filled completely, if possible; thus, each replenishment lot forms at most one partial stack and/or one partial row. We assume replenishment is completed before the beginning of business hours of the next day.

Unit loads are withdrawn by demand during business hours. Unit loads travel from the storage area to the output point and exit the block stacking storage system. In Figure A.2, the retrieval travel from the 10-deep storage area to the output point is depicted. Unit loads are retrieved from

the partial row, when one exists, or from the full row closest to the output point. This operational rule ensures at most one partial row exists for the product lot and increases the likelihood preferred row positions close to the output point are empty.

At the end of business hours, storage locations of product lots are adjusted as necessary. A lot is either relocated, if the assigned storage area for the next day is different, or continues in its present storage area. In Figure A.2, relocation travel from the 8-deep storage area to the 3-deep storage area is illustrated. The possibility of physical conflict between product lots exchanging storage areas is ignored. The relocations are completed before the beginning of the next day.

4. Operating Cost model

In this section, we develop a daily operating cost model of a single product lot. In Section 4.1, we show how to develop the daily space cost model. Space cost is based on the space dedicated for the storage of a single product lot for a day. In Section 4.2, we develop the daily material handling cost model. Material handling cost results from replenishment, retrieval, and relocation; cost calculations are based on the expected travel distance required to move unit loads.

In developing the daily operating cost model, we use the notation in Table A.2. Generally, the inventory level I_t at the end of business hours of day t is equal to the inventory at the beginning of business hours of day t+1; however, when I_t = 0, the inventory level at the beginning of business hours of day t+1 is Q by the assumption of instantaneous replenishment.

The number of rows or the number of required row positions when I_t unit loads of product lot l are stored in d_r-deep storage area, $y_{I_t,r}$, is computed by

$$y_{I_t,r} = \left\lceil \frac{I_t}{d_r z} \right\rceil. \quad \text{(A.1)}$$

Table A.2: Notation for developing DBS cost model

Notation	*Description*
R	set of storage areas considered
q, r	index of storage area at the end of business hours and selected for the next day
d_r	depth of storage area r, measured in unit loads
P_r	number of row positions in a d_r-deep storage area
Q	order quantity of lot, measured in unit loads
I_t	inventory level of lot at the end of business hours of day t, measured in unit loads
$\hat{I}$	inventory level of lot at the latest positioning
D	daily demand of lot, measured in unit loads
z	height of the stack of lot, measured in unit loads
$y_{I_t,r}$	number of rows when I_t unit loads of lot are stored in a d_r-deep storage area
L	length of unit load, measured in feet
W	width of unit load, measured in feet
c	side-to-side clearance between storage rows, measured in feet
H	height of a unit load, including pallet height (if used), measured in feet
A	width of a storage aisle and a cross aisle, measured in feet
v_{hr}	horizontal velocity of a lift truck in a storage row, feet per minute
v_{ha}	horizontal velocity of a lift truck in an aisle, feet per minute
v_v	vertical velocity of a lift truck, feet per minute
c_S	unit floor space cost per square foot per day
c_T	unit material handling cost per minute
M	material handling time of unit load for pick up and put on, measured in minute

We assume all unit loads have the same dimensions and all aisles have the same widths. In addition, we consider clearance between two adjacent rows and no back-to-back clearance between stacks in a row. Figure A.3 depicts unit load, aisle width, and clearance dimensions.

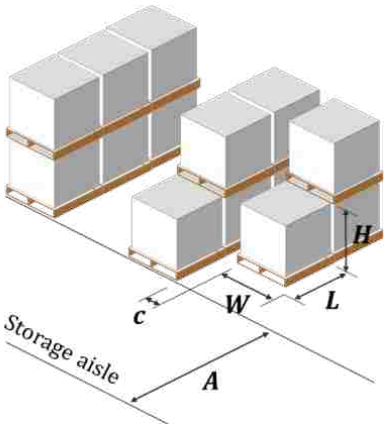

Figure A.3: Dimensions of unit load, width of aisle, and clearance

4.1. Floor space cost

Storage cost is based on floor space. In developing the floor space cost model for a single product lot, we consider the floor space occupied by unit loads, the space loss due to honeycombing, and the aisle space reserved for the row positions assigned. Floor space cost is computed assuming the inventory level at the beginning of business hours; we ignore space savings resulting from decreasing inventory levels during a day.

The occupied floor space can be computed by the number of occupied stack positions or the number of stacks as follows:

$$(W + c)L \left\lceil \frac{I_t}{z} \right\rceil \tag{A.2}$$

where $(W + c)L$ is the planer dimension of the stack position and $\lceil I_t/z \rceil$ is the number of the stacks or the number of occupied stack positions. The dimension of the stack position is illustrated in Figure A.4. Because we assume storage occurs on both sides of the storage aisle, only one-half of the aisle width is "charged" to an occupied storage row.

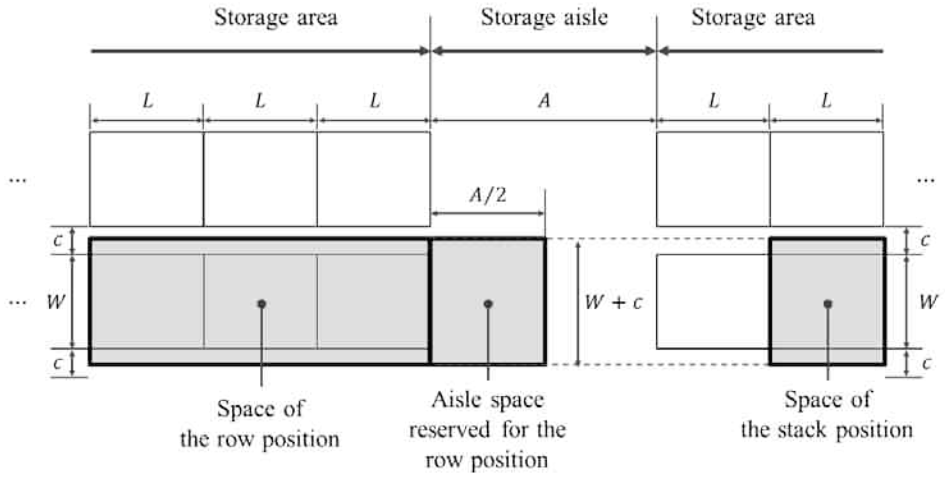

Figure A.4: Dimensions of the row position, the stack position, and the aisle space reserved for the row position

The honeycomb loss can be calculated by the number of unoccupied stack positions in the partial row as follows:

$$(W + c)L\left[d_r - \left(\left\lceil\frac{I_t}{z}\right\rceil \bmod d_r\right)\right] \quad \text{(A.3)}$$

where $d_r - (\lceil I_t/z \rceil \bmod d_r)$ indicates the number of unoccupied stack position in the partial row.

Considering the sum of the number of occupied stack positions and the number of unoccupied stack positions in the partial row can be represented as the product of the row depth and the number of required row positions, the sum of the occupied floor space and the honeycomb loss can be computed by the required number of row position or the number of the rows as follows:

$$(W + c)(d_r L)y_{I_t,r} \quad \text{(A.4)}$$

where $(W + c)(d_r L)$ is the dimension of the row position. Dimension of the row position is depicted in Figure A.4. For example, assume a product lot of 30 unit loads is stored in 3-deep storage area and its stack height is 3. In this case, the sum of the occupied floor space and the honeycomb loss is $4(W + c)(3L)$. The occupied floor space and the honeycomb loss is $10(W + c)L$ and $2(W + c)L$, respectively.

The aisle space reserved for the row positions assigned is computed by

$$(W + c)(0.5A)y_{I_t,r} \quad \text{(A.5)}$$

where $(W + c)(0.5A)$ is the dimension of the aisle space reserved for a row position. We assume only one-half of the aisle width is charged to an occupied storage row because the storage aisle serves both sides of row positions. Therefore, it is defined as the half of the aisle space in front of the row position. The dimension of the aisle space reserved for the row position is illustrated in Figure A.4.

Finally, the dedicated floor space for the lot during a day is computed by combining Equation (A.4) and (A.5) as follows:

$$(W + c)(d_r L + 0.5A)y_{I_t,r}. \tag{A.6}$$

Let FS indicates the daily space cost when I_t unit loads of the product lot is stored in d_r-deep storage area. FS is computed by multiplying c_S and the dedicated floor space for the product lot during a day as follows:

$$FS = \begin{cases} c_S(W + c)(d_r L + 0.5A)y_{I_t,r}, & I_t > 0 \\ c_S(W + c)(d_r L + 0.5A)y_{O,r}, & I_t = 0 \end{cases}. \tag{A.7}$$

When the inventory level is zero, the calculation of the floor space cost is based on the order quantity instead of the inventory level.

4.2. Material handling cost

Daily material handling cost is incurred by the operations of replenishment, retrieval, and relocation. For a single product, replenishment cost, retrieval cost, and relocation cost are based on the required working time to handle all unit loads of the product lot in the operation. The developed model in this section computes expected working time of a single unit load. The working time of the product lot is computed by summing the expected working time of each unit load.

In this section, we first develop the travel distance model for a single unit load in Section 4.2.1; in doing so, we measure distance based on the centerlines of the unit load. In Section 4.2.2 and Section 4.2.3, we show how to specify the storage location of unit loads of the product lot and compute the expected working time for the product lot. Section 4.2.4 provides mathematical formulations of the material handling costs.

4.2.1. Travel distance model of single unit load

In performing replenishment, retrieval, and relocation, the unit load is picked up, moved, and put down. In developing the working time model for a single unit load, we assume the time for picking up and putting down a unit load is constant and identical for all unit loads. Thus, the moving time is calculated based on the expected travel distance required to replenish, retrieval, or relocate the unit load. In estimating the expected travel distance, we decompose a unit load's travel into four parts: travel in a stack position, horizontal travel in a row position, horizontal travel in a storage aisle, and horizontal travel in a cross aisle. Travel distance is measured based on the movement of the center of the unit loads.

Travel in stack position

In a stack, unit loads move vertically between the center of the ground-floor-storage slot and the center of a εth-floor-storage slot. Figure A.5 illustrates the travel of a unit loads between the center of the ground-floor-storage slot and the center of the fourth-floor-storage slot. Let tdS_ε indicate the distance between the center of the ground-floor-storage slot and the center of the εth-floor-storage slot. Then, tdS_ε is computed by

$$tdS_\varepsilon = (\varepsilon - 1)H. \tag{A.8}$$

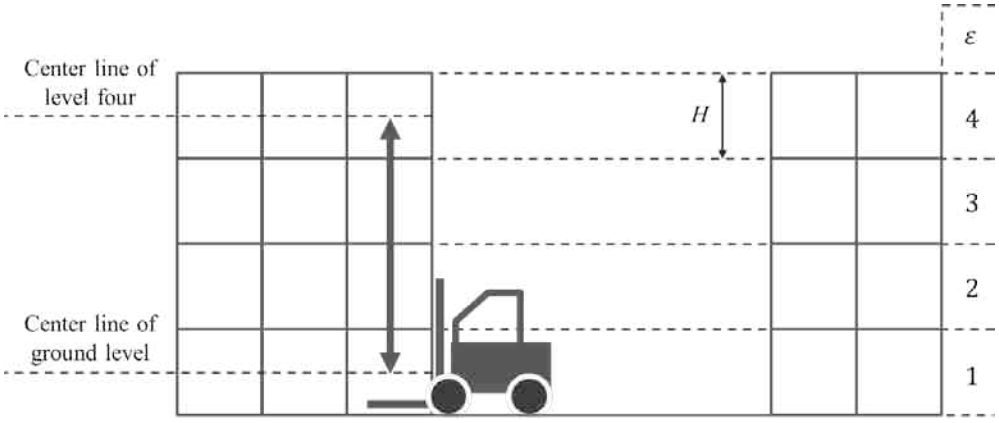

Figure A.5: Vertical travel of unit load in a stack position

Travel in row position

In a row position, unit loads travel horizontally between the entrance of the row position and the center of a stack position. Figure A.6 depict the travel between the entrance of row position 6 and the center of the stack position 5 in the 5-deep storage area. Let η be the stack position and tdR_η be the horizontal distance between the entrance of row position and stack position η, respectively. Then, tdR_η is computed by

$$tdR_\eta = (\eta - 0.5)L. \tag{A.9}$$

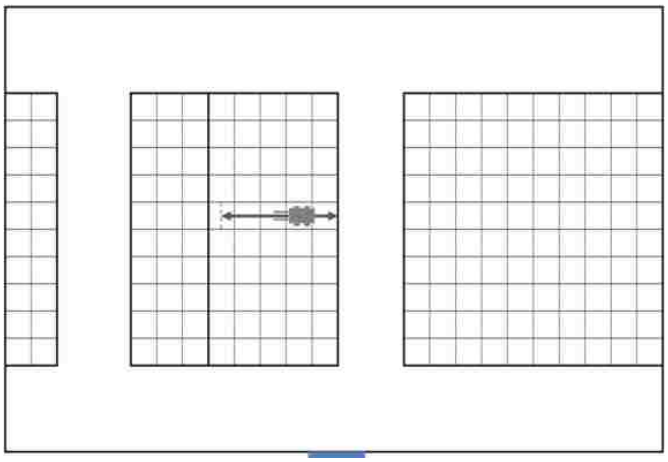

Figure A.6: Horizontal travel of unit load in a row position

Travel in cross aisle

In a cross aisle, the unit load moves horizontally from the input point to the entrance of the storage aisle to perform replenishment, from the entrance of the storage aisle to the output point to perform retrieval, or between the entrances of any two storage aisles.

Figure A.7 illustrates the travel from the input point to the entrance of the storage aisle serving 2-deep and 3-deep storage areas, from the entrance of the storage aisle serving 5-deep and 10-deep storage areas to the output point, and between the entrances of the aisle serving 2-deep and 3-deep storage areas and the aisle serving 5-deep and 10-deep storage areas.

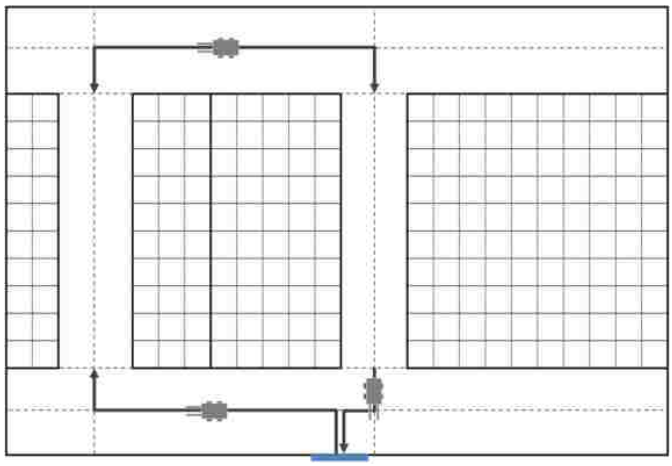

Figure A.7: Horizontal travel of unit load in cross aisle

Let λ and μ be the storage aisle reserved for the d_λ-deep storage area and the d_μ-deep storage area. λ=0 or μ=0 denotes the input/output point. Let $tdC_{\lambda,\mu}$ denotes the horizontal distance in a cross aisle between the entrance of the storage aisle serving the d_λ-deep storage area and the d_μ-deep storage area. Given a layout of the storage system, $tdC_{\lambda,\mu}$ is computed by

$$tdC_{\lambda,\mu} = \|co(\lambda) - co(\mu)\|_1 + A \tag{A.10}$$

where $co(\lambda)$ and $co(\mu)$ are the coordinates of the entrance of the aisle serving the d_λ-deep storage area and the d_μ-deep storage area, respectively, and $\|\alpha - \beta\|_1$ is the L1 norm representing rectilinear distance between two points, α and β.

Travel in storage aisle

In a storage aisle, the unit load travels horizontally between the entrance of a storage aisle and the entrance of a row position. Figure A.8 illustrates the travel between the entrance of a storage aisle accessing the cross aisle of the lower side of the storage system and the entrance of row position 7 in the 3-deep storage area and the travel between the entrance of a storage aisle

accessing the cross aisle of the upper side of the storage system and the entrance of row position 5 in the 5-deep storage area.

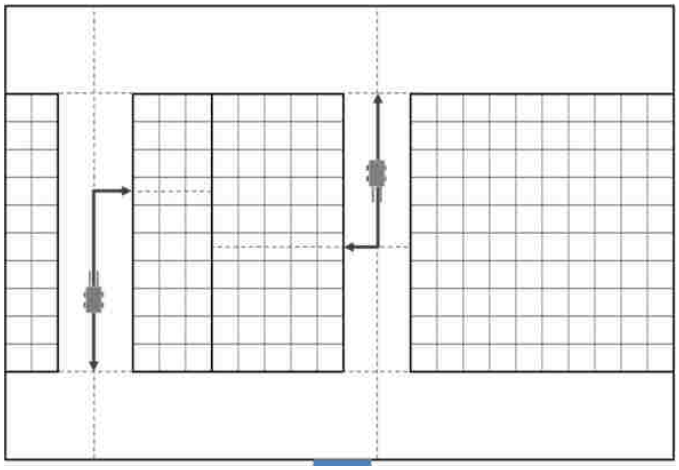

Figure A.8: Horizontal travel of unit load in storage aisle

Let θ be a row position and tdA_θ indicate the horizontal distance in a storage aisle between row position θ and the entrance of the storage aisle accessing the cross aisle of the lower side of the storage system. Then, tdA_θ is computed by

$$tdA_\theta = (\theta - 0.5)(W + c) + 0.5A \tag{A.11}$$

Let $td\hat{A}_\theta$ represent the horizontal distance in a storage aisle between row position θ and the entrance of the storage aisle accessing the cross aisle of the upper side of the storage system. Then, $td\hat{A}_\theta$ is computed by

$$td\hat{A}_\theta = (N - \theta + 0.5)(W + c) + 0.5A. \tag{A.12}$$

4.2.2. Storage location of the unit load

We specify the storage location of the unit load by the storage area, the row position in the storage area, the stack position in the row position, and the level in the stack. When initially positioning the unit load of the product lot during replenishment or relocation, the storage area is determined

first and then row positions are randomly allocated to the lot from among the empty row positions in the storage area, with equally likely probability assigned to all subsets of the empty row positions that are (minimally) large enough to hold all inventory of the product lot. The unit loads are then sequentially located in the row position closest to the input/output point among row positions available, in the deepest stack position available in the row position assigned, and at the lowest level available in the stack position designated. Unit loads are so numbered that the unit load first positioned is referred to as Unit load 1 and the unit load second located is referred to as Unit load 2. Similarly, rows and stacks are numbered; the row and the stack first built are referred to as Row 1 and Stack 1, respectively.

The example of Table A.3 is considered consistently through Section 4.2.2.

Table A.3: Consistent example of Section 4.2.2

Order quantity	6 unit loads
Present inventory level of the product lot	4 unit loads
Stack height of the product lot	2 unit loads
Number of row positions in a storage area	4 row positions

Level of the unit load

Letting γ_u be the storage slot level of Unit load u of the product lot, γ_u is computed by

$$\gamma_u = u - z * \left(\left\lceil \frac{u}{z} \right\rceil - 1\right). \quad \text{(A.13)}$$

In the example of Table A.3, Unit load 1 is positioned at level 1 and Unit load 4 is positioned at level 2.

Stack position of the unit load

Letting $\beta_{u,r}$ be the stack position of Unit load u when the product lot is stored in d_r-deep storage area, $\beta_{u,r}$ can be computed by

$$\beta_{u,r} = d_r \left\lceil \frac{u}{d_r z} \right\rceil - \left\lceil \frac{u}{z} \right\rceil + 1, \tag{A.14}$$

In the example Table A.3, assuming the product lot is stored in a 2-deep storage area, Unit load 1 is located in stack position 2 and Unit load 4 is located in stack position 1.

Row position of the unit load

The row position of a unit load is determined when a product lot is positioned during replenishment or relocation. Because of the randomized rule for positioning a product lot into rows of a given storage area, the row position of a unit load is stochastic.

In this section, we investigate how the expected row position of Unit load u is computed given $\hat{I}$ and I_t at day t, assuming the equally likely empty of row positions. We will not directly use this expected value in our cost model, but that it serves as a convenient way to present some ideas (i.e., expectation over random row subsets) that will be used in our cost model. Notice, $\hat{I}$ is the inventory level of the product lot either (i) immediately after the most recent reorder or (ii) when the product lot was relocated to the d_r-deep storage area, whichever happened more recently. Letting $\alpha_{u,r}$ be the row index of Unit load u when the product lot is stored in a d_r-deep storage area, where $\alpha_{u,r}$ is determined by

$$\alpha_{u,r} = \left\lceil \frac{u}{d_r z} \right\rceil. \tag{A.15}$$

When the product lot is positioned in a d_r-deep storage area, $y_{\hat{I},r}$ row positions are randomly assigned to the product lot. Row 1, …, $y_{I_t,r}$-1 are the full row comprised of $d_r z$ unit loads and Row $y_{\hat{I},r}$ is the partial row comprised of $\hat{I}$-$d_r z(y_{\hat{I},r}$-1) unit loads. A smaller row index is associated with a row position closer to the input/output point.

Let $\{p_1, p_2, \ldots, p_{y_{\hat{I},r}}\}$ be the set of row positions assigned to the product lot among P_r row postions in a d_r-deep storage area. The set has $y_{\hat{I},r}$ elements. Therefore, assuming the P_r row positions are equally likely to be assigned, the probability of the assignment is

$$1/\binom{P_r}{y_{\hat{I},r}}. \tag{A.16}$$

Letting p indicate the row position of $\alpha_{u,r}$ in d_r-deep storage area, the range of p is defined as

$$\alpha_{u,r} \leq p \leq P_r - y_{\hat{I},r} + \alpha_{u,r}. \tag{A.17}$$

The term in the left side of inequality, $\alpha_{u,r}$, represents the index of the closest row position to the input/output point available for Row $\alpha_{u,r}$ in d_r-deep storage area. The term in the right side of inequality, $P_r - y_{\hat{I},r} + \alpha_{u,r}$, expresses the index of the farthest row position from the input/output point available for Row $\alpha_{u,r}$ in d_r-deep storage area.

When Row $\alpha_{u,r}$ is positioned in row position p, Row 1, 2, ⋯, and $\alpha_{u,r}$-1 are stored among row positions 1, 2, ⋯, and p-1; Row $\alpha_{u,r}$+1, $\alpha_{u,r}$+2, ⋯, and $y_{Q,r}$ are accommodated among row positions p+1, p+2, ⋯, and P_r. The number of cases Row 1, 2, ⋯, and $\alpha_{u,r}$-1 are stored among row positions 1, 2, ⋯, and p-1 is computed by $\binom{p-1}{\alpha_{u,r}-1}$ and the number of cases Row $\alpha_{u,r}$+1, $\alpha_{u,r}$+2, ⋯, and $y_{\hat{I},r}$ are accommodated among row positions p+1, p+2, ⋯, and P_r is calculated by $\binom{P_r-p}{y_{\hat{I},r}-\alpha_{u,r}}$. Therefore, the number of storage scenarios where Row $\alpha_{u,r}$ is accommodated in row position p is computed by $\binom{p-1}{\alpha_{u,r}-1}\binom{P_r-p}{y_{\hat{I},r}-\alpha_{u,r}}$.

Given $\hat{I}$, I_t, and d_r for the product lot, assuming row positions are equally likely to be assigned, the expected row position of Unit load $u \leq I_t$ is estimated by

$$\left(\sum_{p=\alpha_{u,r}}^{P_r - y_{\hat{I},r} + \alpha_{u,r}} \binom{p-1}{\alpha_{u,r}-1} \binom{P_r - p}{y_{\hat{I},r} - \alpha_{u,r}} p \right) \Big/ \binom{P_r}{y_{\hat{I},r}} \quad \text{(A.18)}$$

In the example of Table A.3 where $\hat{I}$ = 6, I_t = 4, and z = 2, assume the product lot is stored in a 2-deep storage area. Then, r = 2, d_2 = 2, $y_{6,2}$ = 2, and P_2 = 4. When the product lot is positioned in a 2-deep storage area, two row positions are randomly assigned to the product lot to accommodate six unit loads. Row 1 which is a full row comprised of four unit loads, Unit load 1, 2, 3, and 4 is located in the closer row position among assigned row positions and Row 2 which is a partial row comprised of two unit loads, Unit load 5 and 6 in the farther row position. Table A.4 summarizes all of possible six storage scenarios. The number of all possible scenarios, six, can be computed by $\binom{P_2}{y_{6,2}} = \binom{4}{2}$. Occupying row positions 1 and 2 and 2 and 1 represent the same situation, assuming assigned row positions are filled from the closet one to the input/output point. Possibility of each scenario in the fourth column of the table is computed by Equation (A.16).

Table A.4: Scenario the product lot of Table A.3 occupies two row positions in a 2-deep storage area when replenished

Scenario	Row position		Possibility
	Row 1	Row 2	
1	1	2	1/6
2	1	3	1/6
3	1	4	1/6
4	2	3	1/6
5	2	4	1/6
6	3	4	1/6

In the assignment scenarios, Row 1 is stored in row position 1 three times, row position 2 two times, and row position 3 one time. Thus, Row 1's expected row position is 1.67 computed by {(1*3) + (2*2) + (3*1)}/6. For example, Unit load 3 is a component of Row 1 and thus, its

expected row position is 1.67. Using Equation (A.18)**Error! Reference source not found.**, Unit load 3's expected row position can be computed as follows:

$$\left(\sum_{p=1}^{3}\binom{p-1}{0}\binom{4-p}{1}p\right)\Big/6=\frac{\{(1*3*1)+(1*2*2)+(1*1*3)\}}{6}=\frac{10}{6}\approx 1.67 \qquad \text{(A.19)}$$

4.2.3. Material handling costs of operations

We develop material handling cost models of computing replenishment cost, retrieval cost, and relocation cost of the product lot. Each cost model consists of four sub-models of estimating material handling cost in the stack, in the row, in the storage aisle, and in the cross aisle.

In developing a material handling cost model, two issues arise with regard to preciseness of estimating the cost.

The first issue is an approximation to simplify a mathematical decision making model of the dynamic block stacking problem. A material handling cost model based on the operational assumption of Section 3 and the concept of the expected row position introduced in Section 4.2.2 requires $\hat{I}$ and I_t as an argument. Generally, for mathematical decision making model, the more arguments, the higher dimension of decision variables and consequently the more complexity. Therefore, to reduce the number of arguments, we replace every $\hat{I}$ with I_t in related formulas, approximately estimating material handling cost elements. More details of the approximation are given in following sections.

The second issue is the inconsistency between the assumption that rows are equally likely to be assigned among the row positions in a storage area and the supposed retrieval rule that unit loads are retrieved from the partial row, when one exists, or from the full row closest to the output. The retrieval rule assumption leads to the preference in selecting a row position where a unit load is retrieved and consequently would make row positions unequally likely to be empty.

In cost models of replenishment, retrieval, and relocation, the travel distance of the unit load in a storage aisle is estimated based on the equally likely assumption. Consequently, the material handling cost model bears two contradictory assumptions.

Experimental results of Section 5 demonstrate that the approximation of replacing $\hat{I}$ with I_t and the inconsistency between the equally likely assumption and the supposed retrieval rule do not have a significant effect on assessed costs.

Replenishment cost

Assume the product lot is replenished and a d_r-deep storage area is assigned for storing its unit loads. Letting ttS^{ST} be the total travel time in the stack positions to replenish Q unit loads of the product lot, ttS^{ST} is computed by

$$ttS^{ST} = \left[2 * \sum_{u=1}^{Q} tdS_{\gamma_u}\right] \Big/ v_v. \tag{A.20}$$

Letting ttR^{ST} be the total travel time in the row positions to replenish Q unit loads of the product lot, ttR^{ST} is computed by

$$ttR^{ST} = \left[2 * \sum_{u=1}^{Q} tdR_{\beta_{u,r}}\right] \Big/ v_{vr}. \tag{A.21}$$

Letting ttC^{ST} be the total travel time in the cross aisle to replenish Q unit loads of the product lot, ttC^{ST} is computed by

$$ttC^{ST} = \left[2 * \sum_{u=1}^{Q} tdC_{(0,r)}\right] \Big/ v_{va}. \tag{A.22}$$

Let ttA^{ST} be the expected total travel time in the storage aisle to replenish Q unit loads of the product lot, assuming row positions are equally likely to be empty. It is computed by

$$ttA^{ST} = \left\{ 2 * \sum_{u=1}^{Q} \left[\left(\sum_{p=\alpha_{u,r}}^{P_r - y_{Q,r} + \alpha_{u,r}} \binom{p-1}{\alpha_{u,r}-1} \binom{P_r - p}{y_{Q,r} - \alpha_{u,r}} tdA_{\alpha_{u,r}} \right) \Big/ \binom{P_r}{y_{Q,r}} \right] \right\} \Big/ v_{va}. \quad \text{(A.23)}$$

The terms inside the outer summation represent the expected travel distance of Unit load u based on the assumption rows are equally likely to be assigned among all row positions in the storage area Section 5 shows the effect of the inconsistency between the equally likely assumption and the supposed retrieval rule on estimated costs is inconsiderable.

Letting ST be the daily replenishment cost of a single product lot when a d_r-deep storage area is assigned for storing its unit loads, ST is computed by

$$ST = \begin{cases} 0, & \text{if } I_t > 0 \\ c_T * [ttS^{ST} + ttR^{ST} + ttC^{ST} + ttA^{ST} + (M * Q)], & \text{if } I_t = 0 \end{cases} \quad \text{(A.24)}$$

Retrieval cost

In Section 3, we assume unit loads are retrieved from the partial row, when one exists, or from the full row closest to the output point. To formalize this procedure mathematically, let $U(i)$ denote the index of the unit load retrieved at the ith order, i.e.,

$$U(i) = \begin{cases} \hat{I} - i + 1, & \text{if } 0 < i \leq K \\ \left\lceil \frac{i-K}{d_r * z} \right\rceil * (d_r * z) - \{(i - K - 1) \bmod (d_r * z)\} & \text{if } i > K \end{cases} \quad \text{(A.25)}$$

where $K = \hat{I} \bmod (d_r * z)$ denotes the number of unit loads in the partial row.

In order to derive an estimate of retrieval cost that does not depend on $\hat{I}$, we alter the originally supposed retrieval rule as that unit loads are retrieved from the partial row, when one exists, or from the full row farthest from the output point. This alteration changes Equation (A.25) as

$$U(i) = i \quad \text{(A.26)}$$

where $\hat{I}$ no longer appears. Notice the altered assumption of retrieval rule still contradicts with the assumption that row positions are equally likely to be empty.

For example, consider a product lot of $\hat{I}$ = 10, z = 2, and d_r = 2. Table A.5 summarizes the order based on the original and altered assumption.

Table A.5: Retrieval order of unit loads under the original assumption and the altered assumption

		Retrieval order									
		1	2	3	4	5	6	7	8	9	10
Unit load	Original assumption	10	9	4	3	2	1	8	7	6	5
	Altered assumption	10	9	8	7	6	5	4	3	2	1

Assume the product lot is stored in a d_r-deep storage area and D unit loads among I_t unit loads are retrieved. Letting $ttS_{I_t}^{RT}$ be the total travel time in the stack positions to retrieve D unit loads among I_t unit loads, $ttS_{I_t}^{RT}$ is computed by

$$ttS_{I_t}^{RT} = \left[2 * \sum_{u=I_t-D+1}^{I_t} tdS_{\gamma_u}\right] \Big/ v_v \tag{A.27}$$

Letting $ttR_{I_t}^{RT}$ be the total travel time in the row positions to retrieve D unit loads among I_t unit loads, $ttR_{I_t}^{RT}$ is computed by

$$ttR_{I_t}^{RT} = \left[2 * \sum_{u=I_t-D+1}^{I_t} tdR_{\beta_{u,r}}\right] \Big/ v_{vr} \tag{A.28}$$

Letting $ttC_{I_t}^{RT}$ be the total travel time in the cross aisle to retrieve D unit loads among I_t unit loads, $ttC_{I_t}^{RT}$ is computed by

$$ttC_{I_t}^{RT} = \left[2 * \sum_{u=I_t-D+1}^{I_t} tdC_{(r,0)}\right] \Big/ v_{va} \tag{A.29}$$

Let $ttA_{\hat{I},I_t}^{RT}$ be the expected total travel time in the storage aisle to retrieve D unit loads among I_t unit loads, assuming row positions are equally likely to be empty. It can be computed by

$$ttA_{\hat{I},I_t}^{RT} = \left\{ 2 * \sum_{u=I_t-D+1}^{I_t} \left[\left(\sum_{p=\alpha_{u,r}}^{P_r - y_{\hat{I},r} + \alpha_{u,r}} \binom{p-1}{\alpha_{u,r}-1} \binom{P_r - p}{y_{\hat{I},r} - \alpha_{u,r}} tdA_{\alpha_{u,r}} \right) \Big/ \binom{P_r}{y_{\hat{I},r}} \right] \right\} \Big/ v_{va}. \quad \text{(A.30)}$$

The terms in the outer sigma notation represent the expected travel distance of Unit load u.

Once again, we approximate $ttA_{\hat{I},I_t}^{RT}$ by replacing $\hat{I}$ with I_t in order to simplify the process of computing retrieval costs. Let $\overline{ttA}_{I_t}^{RT}$ be the approximation of $ttA_{\hat{I},I_t}^{RT}$ computed as follows:

$$\overline{ttA}_{I_t}^{RT} = \left\{ 2 * \sum_{u=I_t-D+1}^{I_t} \left[\left(\sum_{p=\alpha_{u,r}}^{P_r - y_{I_t,r} + \alpha_{u,r}} \binom{p-1}{\alpha_{u,r}-1} \binom{P_r - p}{y_{I_t,r} - \alpha_{u,r}} tdA_{\alpha_{u,r}} \right) \Big/ \binom{P_r}{y_{I_t,r}} \right] \right\} \Big/ v_{va}. \quad \text{(A.31)}$$

The terms in the outer summation represent the approximate expected travel distance of Unit load u based on the equally likely assumption that rows are equally likely to be assigned among the set of all row positions in a storage area.

Letting RT be the daily retrieval cost of a single product lot when it is stored in a d_r-deep storage area and D unit loads are retrieved, RT is computed by

$$RT = \begin{cases} c_T * \left[ttS_{I_t}^{RT} + ttR_{I_t}^{RT} + ttC_{I_t}^{RT} + \overline{ttA}_{I_t}^{RT} + (M * I_t) \right], & \text{if } I_t > 0 \\ c_T * \left[ttS_{Q}^{RT} + ttR_{Q}^{RT} + ttC_{Q}^{RT} + \overline{ttA}_{Q}^{RT} + (M * Q) \right], & \text{if } I_t = 0 \end{cases} \quad \text{(A.32)}$$

Relocation cost

Assume the product lot is stored in a d_q-deep storage area and a d_r-deep storage area is newly assigned for the next day. Letting ttS^{BT} be the total travel time in the stack positions to relocate I_t unit loads of the product lot from the d_q-deep storage area to the d_r-deep storage area, ttS^{BT} is computed by

$$ttS^{BT} = \left[4 * \sum_{u=1}^{I_t} tdS_{\gamma_u}\right] \Big/ v_v \tag{A.33}$$

Letting ttR^{BT} be the total travel time in the row positions to relocate I_t unit loads of the product lot from the d_q-deep storage area to the d_r-deep storage area, ttR^{BT} is computed by

$$ttR^{BT} = \left[\left(2 * \sum_{u=1}^{I_t} tdR_{\beta_{u,q}}\right) + \left(2 * \sum_{u=1}^{I_t} tdR_{\beta_{u,r}}\right)\right] \Big/ v_{vr} \tag{A.34}$$

Letting ttC^{BT} be the total travel time in the cross aisle to relocate I_t unit loads of the product lot from the d_q-deep storage area to the d_r-deep storage area, ttC^{BT} is computed by

$$ttC^{BT} = \left[2 * \sum_{u=1}^{I_t} tdC_{q,r}\right] \Big/ v_{va} \tag{A.35}$$

Let $ttA_{\hat{I}}^{BT}$ be the expected total travel time in the storage aisle to relocate I_t unit loads of the product lot from the d_q-deep storage area to the d_r-deep storage area, assuming row positions are equally likely to be empty. It can be computed by

$$ttA_{\hat{I}}^{BT} = \left\{2 * \sum_{u=1}^{I_t} \left[\left(\sum_{p_q=\alpha_{u,q}}^{P_q - y_{\hat{I},q} + \alpha_{u,q}} \sum_{p_r=\alpha_{u,r}}^{P_r - y_{I_t,r} + \alpha_{u,r}} \mathbb{AC}\right) \Big/ \mathbb{B}\right]\right\} \Big/ v_{va} \tag{A.36}$$

where $\mathbb{A}$ is the frequency of cases Unit load u is accommodated in row position p_1 in a d_q-deep storage area and in row position p_2 in a d_r-deep storage area computed by

$$\mathbb{A} = \binom{p_q - 1}{\alpha_{u,q} - 1}\binom{P_q - p_q}{y_{\hat{I},q} - \alpha_{u,q}}\binom{p_r - 1}{\alpha_{u,r} - 1}\binom{P_r - p_r}{y_{I_t,r} - \alpha_{u,r}}, \tag{A.37}$$

$\mathbb{B}$ is the total number of scenarios the product lot occupies $y_{I_t,q}$ row positions among P_q row positions of a d_q-deep storage area and $y_{I_t,r}$ row positions among P_r row positions of a d_r-deep storage area calculated by

$$\mathbb{B} = \binom{P_q}{y_{\hat{I},q}}\binom{P_r}{y_{I_t,r}}, \tag{A.38}$$

and, $\mathbb{C}$ is the sum of the travel distance in the storage aisle serving the d_q-deep storage area and the storage aisle serving the d_r-deep storage area to relocate the product lot from the d_q-deep storage area to the d_r-deep storage area along the shortest path computed by

$$\mathbb{C} = \left|tdS_{\alpha_{u,q}} - tdS_{\alpha_{u,r}}\right| + A, \tag{A.39}$$

if the d_q-deep storage area and the d_r-deep storage area are served by the same storage aisle or

$$\mathbb{C} = \min\left(tdS_{\alpha_{u,q}} + tdS_{\alpha_{u,r}}, td\hat{S}_{\alpha_{u,q}} + td\hat{S}_{\alpha_{u,r}}\right), \tag{A.40}$$

otherwise. In Equation (A.36), the terms in the outer sigma notation represent the expected travel distance of Unit load u.

In order to simplify the relocation cost expression, we approximate $ttA_{\hat{I}}^{BT}$ by replacing $\hat{I}$ with I_t. Let $\overline{ttA}^{BT}$ be the approximation of $ttA_{\hat{I}}^{BT}$ computed as follows:

$$\overline{ttA}^{BT} = \left\{2 * \sum_{u=1}^{I_t}\left[\left(\sum_{p_q=\alpha_{u,q}}^{P_q-y_{I_t,q}+\alpha_{u,q}}\ \sum_{p_r=\alpha_{u,r}}^{P_r-y_{I_t,r}+\alpha_{u,r}} \bar{\mathbb{A}}\mathbb{C}\right)\Big/\bar{\mathbb{B}}\right]\right\}\Big/v_{va} \tag{A.41}$$

where

$$\bar{\mathbb{A}} = \binom{p_q - 1}{\alpha_{u,q} - 1}\binom{P_q - p_q}{y_{I_t,q} - \alpha_{u,q}}\binom{p_r - 1}{\alpha_{u,r} - 1}\binom{P_r - p_r}{y_{I_t,r} - \alpha_{u,r}}, \tag{A.42}$$

and

$$\bar{\mathbb{B}} = \binom{P_q}{y_{I_t,q}}\binom{P_r}{y_{I_t,r}}. \tag{A.43}$$

In Equation (A.41), the terms in the outer summation represent the approximate expected travel distance of Unit load u based on the equally likely assumption that rows are equally likely to be assigned among the set of row positions in a storage area.

Let BT indicate the daily relocation cost of a single product lot when I unit loads of the product lot are relocated from the d_q-deep storage area to the d_r-deep storage area. Then, BT is computed by

$$BT = \begin{cases} c_T * [ttS^{BT} + ttR^{BT} + ttC^{BT} + \overline{ttA}^{BT} + (M * I_t)], & \text{if } I_t > 0 \text{ and } q \neq r \\ 0, & \text{if } I_t = 0 \text{ or } q = r \end{cases} \quad \text{(A.44)}$$

4.3. Daily operating cost model of single product lot

Let OC be the daily operating cost of the single product lot. It is computed by

$$OC = FS + ST + RT + ST \quad \text{(A.45)}$$

In decision making problems, OC is computed for each product lot in advance and given as a parameter.

5. Evaluation of Daily Operating Cost Model

In this section, we evaluate the daily operating cost model developed in this chapter by comparing the cost estimated by the model against the actual cost computed from the realized data in a simulation. The result of the comparison shows how significant the effect of the approximation of replacing $\hat{I}$ with I_t to simplify a mathematical decision making model of the dynamic block stacking problem and the inconsistency between the assumption that row positions are equally likely to be empty and the supposed retrieval rule is on the estimated cost.

At first, we solved BSMPwRuDDs of 135 instances of Group 1 and 135 instances of Group 2 described in Section 5 of CHAPTER 2, establishing their optimal solutions and total costs estimated over a time horizon. Notice the expected total cost for each instance was computed by the daily operating cost model. Based on the optimal solution, the storage area assignment schedule is fixed for each instance.

Then, we conducted simulation experiment using a discrete-event simulation emulating the daily operations of the block stacking system as assumed in Section 3. In the simulation, before

starting daily operation: (i.e., immediately after a product lot is reordered or relocated) product lots' inventory levels and current storage areas are given. The storage area is assigned to each product lot according to the predetermined storage area assignment schedule. Once storage area is determined for each product lot, unit loads' storage-slot levels and stack positions are deterministically specified and row positions are randomly selected among the available row positions in the specified storage area. The operating cost of the following day is computed based on the unit loads' storage positions. This procedure is repeated before starting daily operation over a time horizon of the simulation. At the termination, we have the total cost computed from the realized data in a simulation.

For the evaluation, we compared the cost estimated by the daily operating cost model (i.e., as specified in Section 4) to the cost computed in the simulation experiment (i.e., as assumed in Section 3). Table A.6 summarizes features differentiating the computed cost and the estimated cost. For the purposes of specifying a cost model that is independent of $\hat{I}$, the retrieval rule is altered and the expected travel distance of the unit load in a storage aisle is approximated by replacing $\hat{I}$ with I_t in the formulas. Notice for the estimated cost, the retrieval rule and the assumption that rows of a product lot are equally likely to occupy any subset of rows in a storage area are inconsistent.

Table A.6: Features of total operating cost over a time horizon computed based on data of simulation and estimated by the daily operating cost model

	Computed cost	Estimated cost
Retrieval rule	▪ Unit load are retrieved from the partial row, when one exists, or from the full row closest to the output point	▪ Unit load are retrieved from the partial row, when one exists, or from the full row farthest from the output point
Travel distance of unit load in storage aisle	▪ Travel distance computed based on specific row position defined in simulation	▪ Approximately estimated travel distance, assuming row positions are equally likely to be empty

Considering the randomness in assigning row positions to a product lot, a simulation of a single instance is replicated ten times with fixed storage area assignment schedule. Consequently, data collected from 2700 simulations is used to evaluate the daily operating cost model.

Table A.7 summarizes the absolute gap between the computed cost and the estimated cost calculated by |the computed cost – the estimated cost| / the estimated cost. The third column shows the average absolute gap between the computed total cost and the estimated total cost is very small even though the average absolute gap between the computed retrieval cost and the estimated retrieval cost in sixth column and the computed relocation cost and the estimated relocation cost in the seventh column are considerable.

Table A.8 summarizes the relative gap between the computed cost and the estimated cost calculated by (the computed cost – the estimated cost) / the estimated cost. The third column shows the estimated total cost tends to be overestimated compared to the computed total cost. The fifth and sixth column represent the estimated replenishment cost and retrieval cost are likely to be overestimated compared to the computed replenishment cost and retrieval cost, respectively. The seventh column indicates the estimated relocate cost tends to be underestimated compared to the computed relocation cost.

Table A.7: Summary of the absolute gap between total operating cost over a time horizon computed based on data of simulation and estimated by the daily operating cost model

Number of Lots	Number of Simulations	Ave. absolute gap of total cost (%)	Ave. absolute gap of floor space cost (%)	Ave. absolute gap of replenishment cost (%)	Ave. absolute gap of retrieval cost (%)	Ave. absolute gap of relocation cost (%)
10	450	0.60	0.00	0.46	8.67	3.76
15	450	0.56	0.00	0.63	7.65	3.93
20	450	0.49	0.00	0.50	6.75	3.68
30	450	1.16	0.00	1.17	13.86	6.66
40	450	1.04	0.00	1.00	12.78	6.61
50	450	0.86	0.00	0.80	11.58	5.82
	2700	0.78	0.00	0.76	10.22	5.08

Table A.8: Summary of the relative gap between total operating cost over a time horizon computed based on data of simulation and estimated by the daily operating cost model

Number of Lots	Number of Simulations	Ave. relative gap of total cost (%)	Ave. relative gap of floor space cost (%)	Ave. relative gap of replenishment cost (%)	Ave. relative gap of retrieval cost (%)	Ave. relative gap of relocation cost (%)
10	450	-0.59	0.00	-0.11	-8.67	3.76
15	450	-0.54	0.00	-0.19	-7.65	3.93
20	450	-0.45	0.00	-0.09	-6.74	3.68
30	450	-1.16	0.00	-1.16	-13.86	6.66
40	450	-1.04	0.00	-0.86	-12.78	6.61
50	450	-0.86	0.00	-0.43	-11.57	5.82
	2700	-0.77	0.00	-0.47	-10.21	5.08

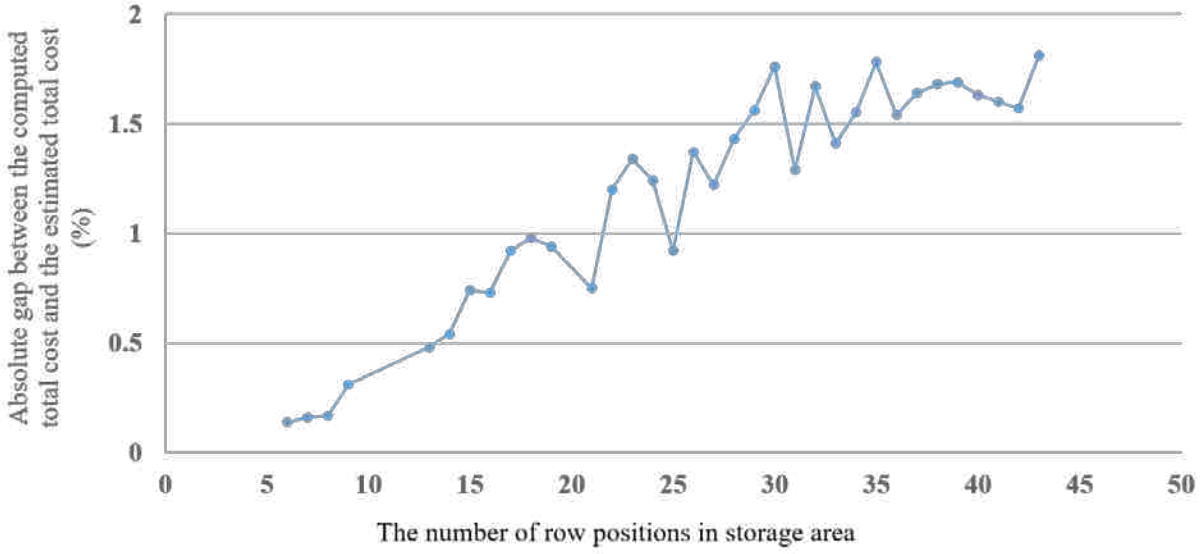

Figure A.9: Change of the absolute gab between the computed total cost and the estimated total cost over increasing number of row positions in the storage area

Figure A.9 shows the absolute gab between the computed total cost and the estimated total cost increases as the number of row position in the storage area increases. However, the optimality gap remains under 2% at the most number of row position. In addition, the optimality gap increases by only 1.67% points as the number of row position increases by 37.

To summarize, the daily operating cost model developed in Section 4 provides an acceptable estimate of the operating cost of block stacking system. Small absolute and relative gap between

the computed cost and the estimated cost in Table A.7 and **Error! Reference source not found.** support this opinion. The observation of Figure A.9 implies the daily operating cost model would give an acceptable estimate for an instance with a large-sized layout.

6. Reference

Bartholdi, J. J., and Hackman S. T. (2014). *Warehouse & Distribution Science*: release 0.96. Georgia Institute of Technology, Atlanta, GA. Retrieved from http://www.warehouse-science.com.

Berry, J. R. (1968). Elements of warehouse layout. *International Journal of Production Research*, 7(2), 105-121.

De Koster, R. (2010). *Warehouse Math*. L. Kroon, T. L, R. Zuidwijk (eds.), Liber amicorum. In memoriam Jo van Nunen, 179-186, Dinalog Breda.

Derhami, S., Smith, J. S., and Gue, K. R. (2017) Optimizing space utilization in block stacking warehouse. *International Journal of Production and Research*, 55(21), 6436-6452.

Goetschalckx, M., and Ratliff, H. D. (1991). Optimal land depth for single and multiple products in block stacking storage systems. *IIE Transactions*, 23(3), 245-258.

Hemmi, J. B. (1963). *An Investigation of the Influence of Warehouse Layout on Storage Costs*. Master's thesis. Georgia Institute of Technology, Atlanta, GA.

Kay, M. G. (2015). *Warehousing*. North Carolina State University, Raleigh, NC. Retrieved from http://www4.ncsu.edu/~kay/Warehousing.pdf.

Kind, D. A. (1965) Measuring warehouse space utilization. *Transportation and Distribution Management*, 5(7), 23-33.

Kind, D. A. (1975). Elements of space management. *Transportation and Distribution Management*, 15(2), 29-34.

Kooy, E. D. (1981). Making better use of available warehouse space. *Industrial Engineering*, 13(10), 26-30.

Larson, T. N., March, H., and Kusiak, A. (1997) A heuristic approach to warehouse layout with class-based storage. *IIE Transactions*. 29(4), 337-348.

Matson, J. O. (1982). *The Analysis of Selected Unit Load Storage Systems*. Doctoral dissertation. Georgia Institute of Technology, Atlanta, GA.

Matson, J. O., Sonnentag, J. J., White, J. A., and Imhoff, R. C. (2014). An analysis of block stacking with lot splitting. In *Proceedings of the 2014 Industrial and System Engineering Research Conference, Y. Guan and H. Liao* (eds), Montreal, Quebec, Canada, 479-506

Moder, J. J., and Thornton, H. M. (1965). Quantitative analysis of the factors affecting floor space utilization of palletized storage. *Journal of Industrial Engineering*, 16(1), 8-18.

Rickles, H. V. and Elliott, K. A. (1985). Spreadsheet programming enables quick custom analyses of material handling problems. Industrial Engineering, 17(2), 80-85.

Roberts, S. D. (1968). *Warehouse Size and Design*. Doctoral dissertation. Purdue University, West Lafayette, IN.

Sonnentag, J. J., Imhoff, R. C., White, J. A., and Matson J. O. (2014) A consideration of the block stacking multi-product storage problem. In *Proceedings of the 2014 Industrial and System Engineering Research Conference, Y. Guan and H. Liao* (eds), Montreal, Quebec, Canada, 3221-3230

Thornton, H. M. (1961). Factors Affecting Space Efficiency of Palletized Storage. Master's thesis. Georgia Institute of Technology, Atlanta, GA.

White, J. A., Sonnentag, J. J., and Matson J. O. (2013). New insights regarding block stacking. In *Proceedings of the 2013 Industrial and System Engineering Research Conference, A Krishnamurthy and W.K.V. Chan* (eds), San Juan, Puerto Rico, 1225-1234.

APPENDIX B. The Dynamic Block Stacking Problem with Random Demand[1]

Hueon Lee, Shengfan Zhang, John A. White

Abstract

The block stacking problem involves determining the depth of a storage row for unit loads to minimize the sum of space and travel costs. A conventional block stacking problem assumes static row depths and deterministic demand. We remove these constraints and treat the dynamic block stacking problem with random demand where row depths are changed (by relocating product). Row depth for a product lot is chosen at periodically designated points in time, based on inventory level and occupied row depth. Between successive epochs (points in time), unit loads are stored in storage rows at the selected depth. At each decision point, the row depth that minimizes total expected discounted cost is chosen. Using a discrete-time infinite-horizon Markov decision process model, the optimal row depth is determined for each inventory level and each occupied row depth. Our findings regarding the number of row depths and the number of relocations of a storage lot during its life provide useful information for warehouse designers.

Keyword

Block stacking, dynamic row depth, random demand, Markov decision process model

[1] APPENDIX B is a reprint of "The Dynamic Block Stacking Problem with Random Demand" in *Proceedings of the 2016 Industrial and System Engineering Research Conference, Institute of Industrial and Systems Engineers*

1. Introduction

Because block stacking is a commonly used storage system, we forgo describing block stacking and reviewing the research literature. For those interested in learning more about block stacking, see Bartholdi and Hackman (2011) and Tompkins et al. (2010), among others; likewise, for fairly recent reviews of the research literature, see Goetschalckx and Ratliff (1991) and Matson et al. (2014).

Suppose 500 unit loads of a particular product are received at a warehouse having available block stacking storage areas of following depths: 2-deep, 3-deep, 5-deep, 10-deep, 15-deep, and 20-deep, as depicted in. Using the solution method given in White, Sonnentag, and Matson (2013), based on unit load, warehouse, and other parameters, results in the product stored in 15-deep storage area. However, if demand is not deterministic, might relocating remaining inventory to a shorter-row-deep storage area prove advantageous? If so, when and how often should relocations occur? We attempt to provide a way to answer these questions in the paper.

With block stacking, floor space is required for a storage row that is not filled completely. Three-dimensional space within a storage row that cannot be used to store another product is called honeycomb loss and the deeper the row depth, the greater the amount of honeycomb loss. At the same time, aisle space is required for each storage row and the shorter the row depth, the greater the percentage of overall space devoted to aisles. Hence, in determining the best row depth for block stacking, one is balancing honeycomb loss and aisle space percentage.

Relocating inventory can increase space utilization, but it increases material handling cost. Each unit load must be picked up and put down when moved to a new storage location and it must be picked up and put down when satisfying a demand for the product. Extra handling means extra cost. However, if relocated closer to the I/O point, it can reduce retrieval travel cost. In the end, if the benefit resulting from changing row depths is greater than the relocation cost,

the lot should be relocated. Unlike the conventional block stacking problem in which an optimal row depth is determined in designing a storage system, the objective of this paper is the development of an optimal relocation policy for a given layout of a block stacking storage system, such as depicted in Figure B.1.

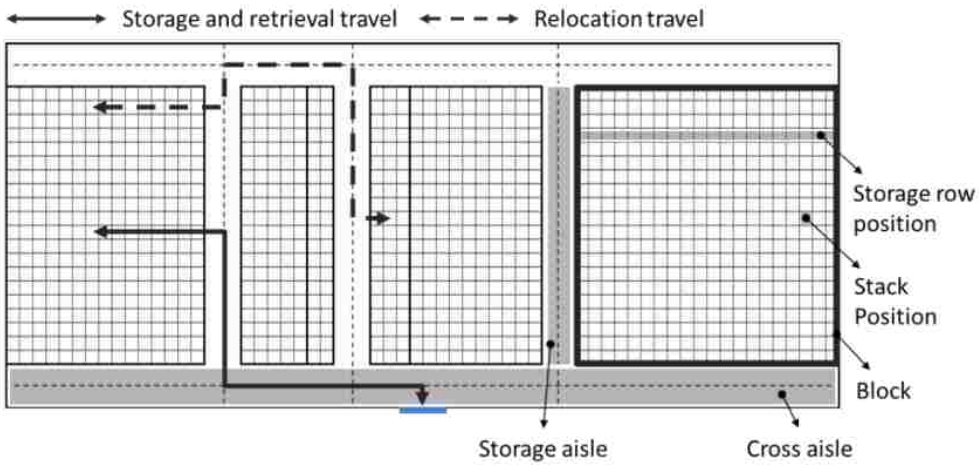

Figure B.1: Block stacking layout for a warehouse

To distinguish the relocation problem from the conventional block stacking problem, we call it the Dynamic Block Stacking (DBS) problem. We believe we are the first to study DBS problem. We develop a cost function for DBS problem including space cost and material handling cost and propose a solution procedure for DBS problem of a single product lot. We assume the number of units demanded during period t is statistically independent of the number of unit loads demanded during period t-1. Based on the following operating procedure, a Markov Decision Process (MDP) model is employed:

- Inventory level of a product lot is checked periodically at the end of business hours.
- Then, the product lot is assigned to a storage area based on the current inventory level.
- If the current storage area and the assigned storage area are different, the product lot is relocated before the beginning of the next business hours.

- If the inventory level is zero, the product lot is reordered and replenished before the beginning of the next business hours.

The paper is organized as follows: DBS problem is modeled as a MDP model; next, a cost function, called the reward function in the MDP model, is developed; then, we provide a numerical example and its optimal relocation policy; and, finally, conclusions are drawn and future research topics are recommended.

2. Markov Decision Process Model for DBSP

A MDP model is a sequential decision making model represented by five key elements: decision epochs, system states, actions, rewards, and state transition probabilities, defined as follows:

Decision epoch

In this research, decision epoch t is a point in time between the end of business hours and the beginning of non-business hours. The time between successive decision epoch t and t+1 is referred to as period t. At a decision epoch, the storage area for a product lot is determined and, if required, unit loads are replenished or relocated immediately. Unit loads are retrieved during a period based on the demand distribution.

System state

State (i_t^l,r_t^l) indicates the inventory level and the index of the storage area of product lot l at decision epoch t. When a given layout of block stack system has R storage areas, r_t^l is a positive integer between one and R. Let d_{r_t} represent the depth of the storage area r_t measured in unit loads. Assuming a product lot is replenished only when the inventory level is zero, i_t^l=0 instantly changes into i_t^l=Q^l where Q^l is the order quantity of product lot l.

Action

Action a_t^l indicates the index of the assigned storage area to product lot l at decision epoch t. The value of a_t^l is a positive integer between one and R. Like d_{r_t}, d_{a_t} represents the depth of the storage area r_t measured in unit loads. If $a_t^l \neq r_t^l$, the product lot is relocated instantly. Note that $a_t^l = r_{t+1}^l$ is always satisfied.

Reward

Reward, $c\left(i_t^l, r_t^l, a_t^l\right)$ is the expected cost incurred by product lot l as the result of taking action a_t^l in state $\left(i_t^l, r_t^l\right)$ at decision epoch t and is calculated based on required floor space and expected material handing for period t. It is dealt with in the following section.

Transition probability

Transition probability, $p\left(i_{t+1}^l, r_{t+1}^l \middle| i_t^l, r_t^l, a_t^l\right)$ is established based on the inventory transition probability derived from the demand distribution. If $a_t^l = r_{t+1}^l$, the transition probability equals the inventory transition probability from i_t^l to i_{t+1}^l. If $a_t^l \neq r_{t+1}^l$, the transition probability is zero. Due to the replenishment assumption, the inventory transition probability at $i_t^l = 0$ is based on $i_t^l = Q^l$.

Solving the sequential decision making problem, we determine the optimal policy prescribing which action is to be taken in any possible future state to maximize expected total reward (or minimize expected total cost). In this paper, with the assumption the reward and transition probabilities do not vary from decision epoch to decision epoch, we model DBSP as an infinite-horizon MDP and determine a stationary optimal policy independent of the decision epoch. For more details regarding MDP, see Puterman (2005).

3. Cost Function

For clarification, a storage row and a stack refer to a measure of unit loads organized in a specified manner; a storage row position and a stack position indicate a reserved storage location for a storage row in a block and for a stack in a storage row, respectively. For convenience, unit loads are numbered sequentially in a lot from the unit load first stored; storage rows of a lot are numbered sequentially in the lot from the storage row first built; storage row positions are numbered sequentially in a block from the row position closest to the I/O point of the system; and, stack positions are numbered sequentially in a storage row from the stack position closest to the storage aisle reserved for the row. To facilitate the development of the cost function, we use the notation of Table B.1 extended from Tompkins et al. (2010) and Matson et al. (2014).

Table B.1: Notations of the cost function of MDP

Notation	*Description*
L	length of a unit load
W	width of a unit load
H	height of a unit load, including the pallet (if used)
c	side-to-side clearance between rows
A	Storage aisle width
z	height of stack, measured in unit load
x	depth of row, measured in unit load
n	number of remaining unit loads of product lot
$y_{n,x}$	number of row positions required to store n unit loads of product lot $$y_{n,x} = \lceil n/xz \rceil$$
$\alpha_{n,x}$	number of full stacks in the last storage row of product lot $$\alpha_{n,x} = \left\lfloor \frac{n - xz(y_{n,x} - 1)}{z} \right\rfloor$$
$\beta_{n,x}$	number of unit loads in the partial stack in the last storage row of product lot $$\beta_{n,x} = n - xz(y_{n,x} - 1) - z\alpha_{n,x}$$
$\gamma_{n,x}$	stack position of the last unit load of product lot $$\gamma_{n,x} = \begin{cases} x - \alpha_{n,x} + 1, & \text{if } \beta_{n,x} = 0 \\ x - \alpha_{n,x}, & \text{if } \beta_{n,x} > 0 \end{cases}$$

Table B.1: Notations of the cost function of MDP (Continue)

Notation	*Description*
$\delta_{n,x}$	level of the last unit load of product lot $\delta_{n,x} = \begin{cases} z, & \text{if } \beta_{n,x} = 0 \\ \beta_{n,x}, & \text{if } \beta_{n,x} > 0 \end{cases}$
S	required floor space during a period
ttS	expected replenishment travel time
$E[ttR, D^l]$	expected retrieval travel time given expected demand D^l
ttB	expected relocation travel time
M	material handling time per a unit loads, including pick up time, put down time, and so on
c_S	floor space cost per square foot per day
c_T	material handling cost per minute
FS	expected floor space cost between consecutive decision epochs
ST	expected replenishment travel cost between consecutive decision epochs
RT	expected retrieval travel cost between consecutive decision epochs
BT	expected relocation travel cost between consecutive decision epochs
v_v	vertical velocity of the lift truck in a stack
v_{hr}	horizontal velocity of the lift truck in a row
v_{ha}	horizontal velocity of the lift truck in a storage aisle
tdS_i	travel distance in stack from ground to the bottom of the level i, (i-1)H
$tdC_{i,j}$	travel distance in cross aisle between the entrance of the aisle reserved for the storage area i and the entrance of the aisle reserved for storage area j, measured in a given layout (i=0 or j=0 indicates the I/O point of the system)
tdA_i	travel distance in storage aisle between the entrance of the aisle and the entrance of the ith row position, (i-0.5)(W+c)+0.5A
tdR_i	travel distance in storage row between the entrance of the row position and the centerline of the ith stack position, (i-0.5)L
N	number of rows of block, same for each block

3.1. Floor space cost

Reserved floor space for a lot at a decision epoch and during the following period is

$$S = y_{i_t^l, a_t^l}(W + c)\left(a_t^l L + 0.5A\right). \tag{B.1}$$

Hence, if action a_t^l is chosen at state (i_t^l, r_t^l), expected floor space cost between consecutive decision epochs is

$$C^{FS}\left(i_t^l, a_t^l\right) = c^s S. \tag{B.2}$$

3.2. Travel cost

In DBSP, we include three travel components: storage travel, retrieval travel, and relocation travel. To calculate expected travel times, we decompose each trip into vertical travel in a stack, horizontal travel in a cross aisle, horizontal travel in a storage aisle, and horizontal travel in a storage row. To clarify the calculation, we assume a round trip; a storage row position is filled from the farthest stack position (from-the back-to-the front); in a block, each storage row position has the same probability to be empty; an *a*-deep block selected at a decision epoch has sufficient empty row positions for replenishment and relocation; unit loads are retrieved from the last stored load (last-in-first-out in a lot); for relocation travel, the shortest path is always used; and lot splitting is not allowed.

Replenishment travel cost

Consider expected vertical travel distance in a full stack and in a partial stack. Given the number of full stacks of the lot replenished, $\left(y_{Q^l,a_t^l}-1\right)a_t^l+\alpha_{Q^l,a_t^l}$, total vertical travel distance in stacks, td^{st}, is

$$td^{st}=2\left\{\sum_{j=1}^{z} d_j^{st}\left[\left(y_{Q^l,a_t^l}-1\right)a_t^l+\alpha_{Q^l,a_t^l}\right]+\sum_{j=1}^{\beta_{Q^l,a_t^l}} d_j^{st}\right\}. \tag{B.3}$$

Similarly, consider expected horizontal travel distance in a full storage row of *a*-depth and in a partial row. Given the number of full storage rows of the lot replenished, $y_{Q^l,a_t^l}-1$, total horizontal travel distance in storage rows, td^{sr}, is

$$td^{sr}=2\left\{z\sum_{j=1}^{a_t^l} d_j^{sr}\left(y_{Q^l,a_t^l}-1\right)+\left[z\sum_{j=\gamma_{Q^l,a_t^l}}^{a_t^l} d_j^{sr}-\left(z-\delta_{Q^l,a_t^l}\right)d_{\gamma_{Q^l,a_t^l}}^{sr}\right]\right\}. \tag{B.4}$$

Total horizontal travel distance in cross aisles, td^{ca}, is $2Qd_{0,a}^{ca}$. We cannot specify which storage row positions will be occupied by storage rows of a lot. Thus, we consider all possible scenarios in which y_{Q^l,a_t^l} row positions are selected from among N row positions. It gives $C\left(N, y_{Q^l,a_t^l}\right)$ cases. Notice, the storage row position occupied by the jth storage row is closer than the (j+1)th storage row. Available storage row positions for the jth storage row are from the jth position to the $\left[N - y_{Q^l,a_t^l} + j\right]$th position. The number of scenarios where the jth storage row occupies the kth position equals $C(k\text{-}1, j\text{-}1)C\left(N\text{-}k, y_{Q^l,a_t^l}\text{-}j\right)$. Thus, the expected horizontal travel distance in a storage aisle for the jth storage row is

$$2 \sum_{k=j}^{N-y_{Q^l,a_t^l}+j} \binom{k-1}{j-1}\binom{N-k}{y_{Q^l,a_t^l}-j} d_k^{sa} \Big/ \binom{N}{y_{Q^l,a_t^l}}. \tag{B.5}$$

Since the jth unit load is in the y_{j,a_t^l}th storage row, total horizontal travel distance in a storage aisle, td^{sa}, is

$$td^{sa} = 2\sum_{j=1}^{Q^l}\left[\sum_{k=y_{j,a_t^l}}^{N-y_{Q^l,a_t^l}+y_{j,a_t^l}} \binom{k-1}{y_{j,a_t^l}-1}\binom{N-k}{y_{Q^l,a_t^l}-y_{j,a_t^l}} d_k^{sa} \Big/ \binom{N}{y_{Q^l,a_t^l}}\right]. \tag{B.6}$$

Storage travel time, T^{ST}, and storage travel cost, $C^{ST}\left(i_t^l, a_t^l\right)$, are

$$T^{ST} = \left(\frac{td^{st}}{v_v}\right) + \left(\frac{td^{sr}}{v_{hr}}\right) + \left(\frac{td^{ca} + td^{sa}}{v_{ha}}\right) \tag{B.7}$$

and

$$C^{ST}\left(i_t^l, a_t^l\right) = \begin{cases} c^t(T^{ST} + H * Q), & \text{if } i_t^l = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{B.8}$$

Retrieval travel cost

In calculating retrieval travel cost, the demand distribution is considered. Assuming demand is *D*, total vertical travel distance in stacks, td_D^{st} and total horizontal travel distance in storage rows, td_D^{sr} are

$$td_D^{st} = 2 \sum_{j=i_t^l - D^l + 1}^{i_t^l} d_{\delta_{i_t^l, a_t^l}}^{st}, \tag{B.9}$$

and

$$td_D^{sr} = 2 \sum_{j=i_t^l - D^l + 1}^{i_t^l} d_{\gamma_{i_t^l, a_t^l}}^{sr} \tag{B.10}$$

Total horizontal travel distance in cross aisles, td_D^{ca}, is $2D^l d_{0,a_t^l}^{ca}$. Total horizontal travel distance in a storage aisle, $td_{D^l}^{sa}$, is

$$td_{D^l}^{sa} = 2 \sum_{j=i_t^l - D^l + 1}^{i_t^l} \left[\sum_{k=y_{i_t^l, a_t^l}}^{N - y_{Q^l, a_t^l} + y_{i_t^l, a_t^l}} \binom{k-1}{y_{i_t^l, a_t^l} - 1} \binom{N-k}{y_{Q^l, a_t^l} - y_{i_t^l, a_t^l}} d_k^{sa} \Big/ \binom{N}{y_{Q^l, a_t^l}} \right] \tag{B.11}$$

Given demand D^l, retrieval travel time, $T_{D^l}^{RT}$, is

$$T_{D^l}^{RT} = \left(\frac{td_{D^l}^{st}}{v_v} \right) + \left(\frac{td_{D^l}^{sr}}{v_{hr}} \right) + \left(\frac{td_{D^l}^{ca} + td_{D^l}^{sa}}{v_{ha}} \right) \tag{B.12}$$

By the relation between the demand distribution, inventory transition probability, and state transition probability, the probability of demand *D* can be represented by the state transition probability as follows:

$$\Pr(\text{Demand=D}) = p\left(i_t^l - D^l, a_t^l \middle| i_t^l, r_t^l, a_t^l\right). \tag{B.13}$$

Hence, retrieval travel cost, $C^{RT}\left(i_t^l, a_t^l\right)$, is

$$C^{RT}\left(i_t^l, a_t^l\right) = \sum_{D^l=1}^{i_t^l} \left[c^t\left(T_{D^l}^{RT} + MH * D\right) p\left(i_t^l - D^l, a_t^l \middle| i_t^l, r_t^l, a_t^l\right)\right]. \tag{B.14}$$

Relocation travel cost

In the calculation of relocation travel cost, we consider travel in the $d_{r_t^l}$-deep storage area and in the $d_{a_t^l}$-deep storage area, separately. Similar to the calculation of replenishment travel distance, total vertical travel distance in stacks, td^{st}, is

$$\begin{aligned} td^{st} = 2\left\{\sum_{j=1}^{z} d_j^{st}\left[\left(y_{i_t^l, r_t^l} - 1\right) r_t^l + \alpha_{i_t^l, r_t^l}\right] + \sum_{j=1}^{\beta_{i_t^l, r_t^l}} d_j^{st}\right\} \\ + 2\left\{\sum_{j=1}^{z} d_j^{st}\left[\left(y_{i_t^l, a_t^l} - 1\right) r_t^l + \alpha_{i_t^l, a_t^l}\right]\right\} + \sum_{j=1}^{\beta_{i_t^l, a_t^l}} d_j^{st}, \end{aligned} \tag{B.15}$$

and total horizontal travel distance in storage rows, td^{sr}, is

$$\begin{aligned} td^{sr} = 2\left\{z\sum_{j=1}^{r_t^l} d_j^{sr}\left(y_{i_t^l, r_t^l} - 1\right) + \left[z\sum_{j=\gamma_{i_t^l, r_t^l}}^{r_t^l} d_j^{sr} - \left(z - \delta_{i_t^l, r_t^l}\right) d^{sr}_{\gamma_{i_t^l, r_t^l}}\right]\right\} \\ + 2\left\{z\sum_{j=1}^{a_t^l} d_j^{sr}\left(y_{i_t^l, a_t^l} - 1\right) + \left[z\sum_{j=\gamma_{i_t^l, a_t^l}}^{a_t^l} d_j^{sr} - \left(z - \delta_{i_t^l, a_t^l}\right) d^{sr}_{\gamma_{i_t^l, a_t^l}}\right]\right\}. \end{aligned} \tag{B.16}$$

Total horizontal travel distance in cross aisles, td^{ca}, is given by $2i_t^l d^{ca}_{a_t^l, r_t^l}$. Total horizontal travel distance in storage aisles, td^{sa}, is calculated as follows:

$$td^{sa} = 2\sum_{j=1}^{i_t^l}\left[\sum_{k=y_{j,r_t^l}}^{N-y_{i_t^l, r_t^l}+y_{j,r_t^l}} \sum_{m=y_{j,a_t^l}}^{N-y_{i_t^l, a_t^l}+y_{j,a_t^l}} \frac{\mathbb{A}}{\mathbb{B}}\mathbb{C}\right] \tag{B.17}$$

where

$$\mathbb{A} = \binom{k-1}{y_{j,r_t^l}-1}\binom{N-k}{y_{i_t^l,r_t^l}-y_{j,r_t^l}}\binom{m-1}{y_{j,a_t^l}-1}\binom{N-m}{y_{i_t^l,a_t^l}-y_{j,a_t^l}}, \tag{B.18}$$

$$\mathbb{B} = \binom{N}{y_{i_t^l,r_t^l}}\binom{N}{y_{i_t^l,a_t^l}}, \tag{B.19}$$

and

$$\mathbb{C} = \begin{cases} |k-m|(W+c)+A, & \text{if storage area } r_t^l \text{ and } a_t^l \text{ share the same aisle,} \\ \min\big(k+m, 2N-(k+m)\big)(W+c)+A, & \text{otherwise.} \end{cases} \tag{B.20}$$

Relocation travel time, T^{RL}, and storage travel cost, $C^{RL}\left(i_t^l,r_t^l,a_t^l\right)$, are

$$T^{RL} = \left(\frac{td^{st}}{v_v}\right) + \left(\frac{td^{sr}}{v_{hr}}\right) + \left(\frac{td^{ca}+td^{sa}}{v_{ha}}\right) \tag{B.21}$$

and

$$C^{RL}\left(i_t^l, r_t^l, a_t^l\right) = \begin{cases} c^t\left(T^{RL} + MH * i_t^l\right), & \text{if } r_t^l \neq a_t^l \text{ and } i_t^l \neq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{B.22}$$

3.3. Total cost function

Total cost is the sum of floor space cost, storage travel cost, retrieval travel cost, and relocation travel cost, and is calculated as follows:

$$TC\left(i_t^l, r_t^l, a_t^l\right) = C^{FS}\left(i_t^l, a_t^l\right) + C^{ST}\left(i_t^l, a_t^l\right) + C^{RT}\left(i_t^l, a_t^l\right) + C^{RL}\left(i_t^l, r_t^l, a_t^l\right). \tag{B.23}$$

The DBSP MDP model reward function is the product of minus one and the total cost function:

$$r\left(i_t^l, r_t^l, a_t^l\right) = -1 * TC\left(i_t^l, r_t^l, a_t^l\right). \tag{B.24}$$

4. Case

To demonstrate the solution procedure, consider an example having the following design parameters: Q = 60; L = 4 ft; W = 3.5 ft; c = 0.75 ft; H = 4.5 ft; A = 13 ft; z = 2; v_{ha} = 240 fpm; v_{hr} = 80 fpm; v_v = 50 fpm; c^s = 0.1 \$/ft^2/day; c^t = 0.5 \$/min; MH = 0.5 min/unit load. Although

any discrete probability mass function can be used, for convenience, we assume demand is Poisson distributed with an average demand (λ) of 5 unit loads/day. The storage layout includes 6 storage blocks, each having 20 storage row positions and storage depths shown in Figure 1. To obtain an optimal policy for the DBSP MDP model, we use the value iteration algorithm with the expected total discounted reward optimality criterion described in Puterman (2005). Using a daily discount rate of 0.97, the optimal policy recommends storing the lot in the 10-deep block at replenishment and at inventory levels greater than 8 and storing the remainder of the lot in 2-deep or 3-deep blocks with inventory levels not greater than 8. The resulting expected daily cost is \$48.96. If relocations are not allowed and the product is stored in 10-deep storage rows for the life of the lot, the expected daily cost is \$49.97. Hence, allowing relocation allowed daily cost to be reduced by 2.03%. When c^s = 0.3 \$/ft^2/day, c^t = 0.3 \$/min, and λ = 2 unit loads/day, the optimal policy includes frequent relocations over all blocks with an expected daily cost of \$110.43. Without relocation, using 10-deep storage rows for the life of the lot yields an expected daily cost of \$121.84. Thus, performing relocation during the life of the lot reduces daily cost by 9.36%.

To provide insight regarding the DBSP, we performed a sensitivity analysis by considering several values for three parameters in the example: storage cost, space cost, average demand, and order quantity. Specifically, we solved the example problem with: c^s = 0.1 and 0.5; c^t = 0, 0.05, 0.1, 0.2, …, 1.5; λ = 2, 3, 5, and 10; and Q = 15, 60, 300, and 500. Increased space cost, coupled with decreased travel cost led to frequent relocations with larger inventory levels. Also with c^s = 0.1 \$/ft^2/day and c^t = 1.5 \$/min for a ratio of 15, no relocation occurred in the optimal policy. Changing λ produced changes in the optimal policy, but no discernable pattern was evident. Changing Q yielded the following insights: for very large values of Q, a large number of very

deep storage rows are required, at most one of which will be partially filled and honeycomb loss is a small fraction of the storage space; hence, relocation does not occur until the inventory level reaches a point when honeycomb loss becomes a significant fraction of the space occupied by the product lot. When the cost of relocation is less than the cost of honeycomb loss, relocation occurs. As an example, for $Q = 500$, no relocation occurred until inventory reached a level of 20.

5. Conclusions

In the paper, we proposed changing row depths during the life of a storage lot to reduce the cost of operating a block stacking storage system. The dynamic block stacking problem was formulated and solved using a Markov decision process model. Our computational experience yielded unexpected results. In situations where we expected an optimal policy would include relocations, none occurred. Likewise, when we anticipated no relocations would occur, they did. Based on the computational experience, we gained additional insights regarding combinations of parameter values that produce relocations and combinations that do not produce relocations.

For further research, we believe allowing lot splitting in a multi-product version of the DBSP problem is worthy of consideration. Likewise, we anticipate some interesting design issues will arise when considering the DBSP with deterministic demand. Finally, we are confident an incorporation of relocation considerations in the design of a new block stacking layout will be a challenging endeavor.

6. References

Bartholdi, J. J., III and Hackman, S. T. (2011). *Warehousing & Distribution Science*, Release 0.96, www.warehouse-science.com

Tompkins, J. A., White, J. A., Bozer, Y. A., and Tanchoco, J. M. A. (2010). *Facilities Planning*, 4th Edition, John Wiley & Sons, Inc., New York, NY

Goetschalckx, M. and Ratliff, H. D. (1991). Optimal Lane Depths for Single and Multiple Products in Block Stacking Storage Systems. *IIE Transactions*, 23(3), 245-258.

Matson, J. O., Sonnentag, J. J, White, J. A. and Imhoff, R. C. (2014). An Analysis of Block Stacking with Lot Splitting. *Proceedings of the 2014 Industrial and Systems Engineering Research Conference, T. Guan and H. Liao (eds)*, May 31- June 3, Montreal, Quebec, Canada, 497-506.

White, J. A., Sonnentag, J. J., and Matson, J. O. (2013). *New Insights Regarding Block Stacking. Proceedings of the 2013 Industrial Engineering Research Conference, A. Krishnamurthy an W. K. V. Chan (eds)*, May 30-June 2, San Juan, Puerto Rico, 1225-1234.

Puterman, M. L. (2005). *Markov Decision Processes*, John Wiley & Sons, Inc., Hoboken, NJ