DECISION SCIENCES INSTITUTE

Improving the In-Deep Storage in Flow-Rack Automated Storage and Retrieval Systems

Zhuxi Chen
College of Information Engineering, Yangzhou University
Email: zxchen@yzu.edu.cn

Jatinder N.D. Gupta
College of Business, University of Alabama in Huntsville
Email: guptaj@uah.edu

ABSTRACT

In-deep method, which stores fast-turnover unit-loads in front of slow-turnover unit-loads, is effective for the storage assignment in flow-rack automated storage and retrieval systems with class-based storage policy. In this paper, an adaption of In-deep method and a Modi-InDeep method are proposed. Experimental results demonstrate that the adaption can cut down the costs of warehouse operations by reducing blocking unit-loads compared to In-deep method. The Modi-InDeep method can not only decrease the number of blocking unit-loads but also store more unit-loads to easy accessible bins than In-deep method does to improve the performance of storages and retrievals.

KEYWORDS: class-based, storage assignment, flow-rack, AS/RS

INTRODUCTION

Class-based storage policy, which divides items into different classes according to their turnover rates and stores fast-turnover items to easy accessible zones of warehouse, is widely applied in practice. Normally, zone division (Yu et al. 2015), zone sizing (Guo et al. 2016) and zone positioning (Yu et al. 2009) are three main topics for class-based storage in traditional warehouses and automated storage and retrieval systems (AS/RS).

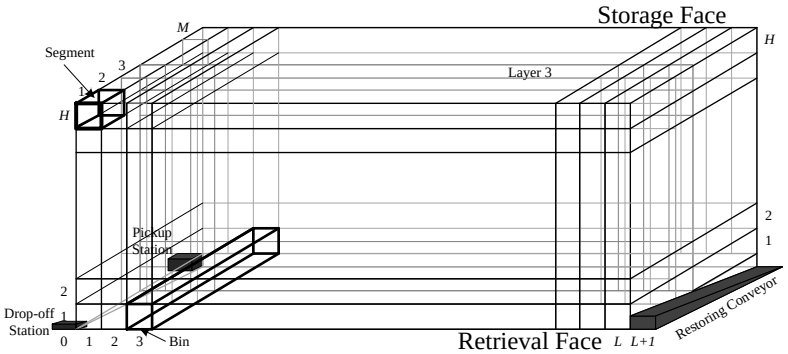
Over the recent decades, high density AS/RSs, including flow-rack (Sari et al. 2005), mobile-rack (Chang et al. 2006), 3D compact (De Koster et al. 2008), dual-deep (Lerher et al. 2010), puzzle-based (Gue et al. 2014), bidirectional flow-rack (Chen et al. 2016) and live-cube (Zaerpour et al. 2017a), have been designed to mitigate the increasing land cost. Because of their configurations, new methods or policies for class-based storage policy must be proposed, such as in live-cube AS/RS (Zaerpour et al. 2017b) and flow-rack AS/RS (Cardin et al. 2012). In-deep method stores fast-turnover unit-loads in front of slow-turnover unit-loads because of the special configuration of flow-rack. Although In-deep method obtains good solution, the class separation should be carefully analyzed because it simply assigns 20% fast-turnover unit-loads into Class-A and the rest unit-loads into Class-B. As well, a novel method should be designed to take the advantage of the best class separation. The rest parts of this paper are problem description, proposed methods, performance evaluation and conclusions.

PROBLEM DESCRIPTION

Figure 1 illustrates the structure of a flow-rack AS/RS. A storage machine in the storage face loads incoming unit-loads from the pick-up station and stores them to appropriate bins.

Correspondingly, a retrieval machine in the retrieval face retrieves outgoing unit-loads from their stored bins and moves them to the drop-off station. Each bin can simultaneously store at most *M* unit-loads. Unit-loads in each bin slide from the storage face to the retrieval face driven by the gravity. Unit-loads in each bin follow first-in-first-out (FIFO) mode. Therefore, an outgoing unit-load may be blocked by blocking unit-loads, which must be removed to the restoring conveyor before retrieving target ones. As well, the storage machine must restore blocking unit-loads to the storage face from the restoring conveyor.

Figure 1. A flow-rack AS/RS with *L* columns and *H* rows of bins and *M* layers



In class-based storage policy, a unit-load can be denoted by a tuple with the form of (*id*, *ref*) where *id* is a unique identifier and *ref* is the reference (type). Assume that there are *N* types of unit-loads denoted as {1, 2, …, *N*}. The demand per unit-time (turnover rate) of each type is constant and known. For the $\boldsymbol{i^{th}}$ type of unit-loads, the annual demand is $\boldsymbol{D(i)}$. Without loss of generality, let $\boldsymbol{D(1) \geq D(2) \ldots \geq D(N)}$ and $\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{D(i)} = \boldsymbol{A}$ where *A* is the annual demand of all unit-loads in a warehouse. Based on the ABC-demand curve, let $\boldsymbol{G(i) = (i/N)^s}$ where $\boldsymbol{0 < s \leq 1}$, $\boldsymbol{i \in \{1, \ldots, N\}}$ and $\boldsymbol{D(i) = A(G(i) - G(i-1))}$. It can be known that the first 20% fast unit-loads contribute 90%, 80%, 70%, 60%, 50%, 40%, 30% and 20% of turnovers when *s* takes value of 0.065, 0.139, 0.222, 0.317, 0.431, 0.569, 0.748 and 1, respectively.

A storage request with form of (*id*, *ref*) represents that a *ref* type unit-load with identifier *id* will be stored. Normally, incoming unit-loads stored in bins close to the pick-up station can reduce the travel times (costs) of storage and retrieval. However, unsuitable stored sequence of unit-loads in a bin brings blocking unit-loads because of the FIFO mode. Therefore, the stored sequence of unit-loads in each bin is more important than the travel time between the pick-up station and the storage bin. A retrieval request with form of (0, *ref*) signifies that a *ref* type unit-load will be retrieved from the flow-rack. Because of the presence of blocking unit-loads, it needs to select an outgoing unit-load with fewest blocking unit-loads from a bin close to the restoring conveyor. As well, the travel time between selected bin and the drop-off station should be carefully considered for minimizing the cost of removing blocking unit-loads. In summary, a storage assignment method for the flow-rack AS/RS with class-based storage policy needs to (1) take full advantage of turnover information to reduce blocking unit-loads and (2) store incoming unit-loads to easy accessible bins for reducing travel time of retrieval.

PROPOSED METHODS

Adaption for In-Deep Method (InDeep_n)

Cardin et al. (2012), proposed the In-deep method. The In-deep method assigns first 20% types of fast-turnover unit-loads into Class-A and the rest 80% types of slow-turnover unit-loads into

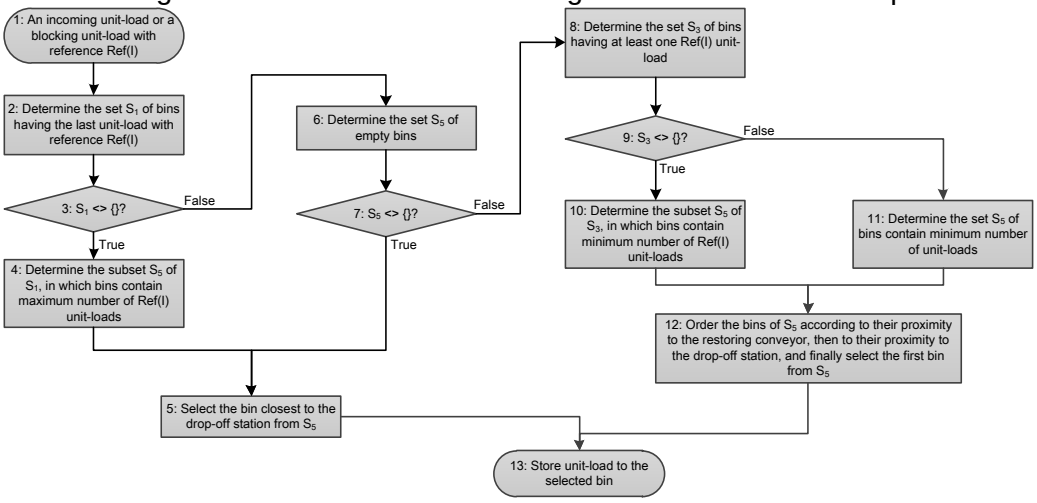
Class-B. The In-deep method consists of six algorithms: (a) priority rule, (b) main framework of storage, (c) storage algorithm for Class-A, (d) storage algorithm for Class-B, (e) retrieval algorithm for Class-A and (f) retrieval algorithm for Class-B. Algorithm (a), (b), (c) and (d) generate storage/restore operations. Algorithm (e) and (f) generate retrieval operations. The main idea of In-deep method is to store fast-turnover unit-loads in front of slow-turnover unit-loads in each bin to reduce the number of blocking unit-loads. Experimental results in Cardin et al. 2012 illustrate that In-deep method significantly reduces blocking unit-loads compared with random storage assignment method.

A simple and intuitive adaption for In-deep method is to alter the class separation of unit-loads. Therefore, InDeep_n is proposed which assigns first *n*% of fast-turnover unit-loads to Class-A and the rest unit-loads to Class-B. Therefore, InDeep_20 is the method proposed in Cardin et al. 2012. Table 1 and 2 in this paper illustrate that InDeep_100, i.e., assign all unit-loads into Class-A, can obtain the lowest average travel times of storage and retrieval operations.

Modified In-deep Method (Modi-InDeep)

Since InDeep_100 method has the best performance among InDeep_n methods, a modified In-deep method (denoted as Modi-InDeep) was designed based on the InDeep_100 method. The storage/restore of Modi-InDeep (shown in Figure 2) merges Algorithms (b) and (c) of the In-deep method. There are four paths start from Stpe 1 to Step 13 as shown in Figure 2. Let P_1 = (1, 2, 3, 4, 5, 13), P_2 = (1, 2, 3, 6, 7, 5, 13), P_3 = (1, 2, 3, 6, 7, 8, 9, 10, 12, 13) and P_4 = (1, 2, 3, 6, 7, 8, 9, 11, 12, 13). Actually, P_1 is modified by Algorithm (b) as well as P_2, P_3 and P_4 are modified by Algorithm (c) of In-deep method. The main difference is that unit-loads follow P_1 and P_2 are stored to a bin closest to the drop-off station because there is no blocking unit-load caused by these two cases. Unit-loads follow P3 and P4 are stored to a bin closest to the restoring conveyor because blocking unit-loads may be caused. Besides storage operation, restoring operation can be generated by this method as well.

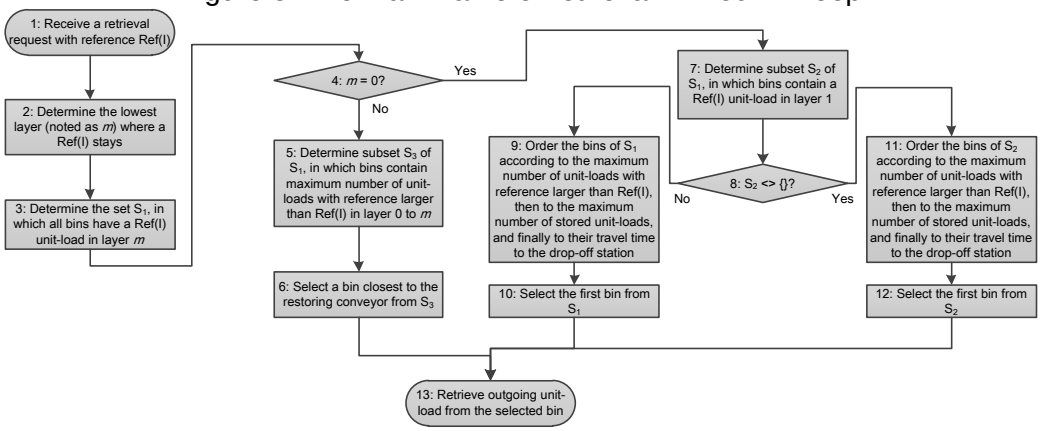
Figure 2. The main frame of storage/restore in Modi-InDeep



Correspondingly, the retrieval of Modi-InDeep (shown in Figure 3) is based on Algorithm (e) of In-deep method. Step 2 ensures that all stored Ref(I) unit-loads with fewest blocking unit-loads are selected as the candidates. There are three paths from Step 1 to Step 13, which denoted as P_5 = (1, 2, 3, 4, 5, 6, 13), P_6 = (1, 2, 3, 4, 7, 8, 9, 10, 13) and P_7 = (1, 2, 3, 4, 7, 8, 11, 12, 13). There is no blocking unit-load when outgoing unit-loads follow P_6 and P_7. In this case,

Modi-InDeep method chooses an outgoing unit-load from a bin closest to the drop-off station. Otherwise, Modi-InDeep method selects an outgoing unit-load from a bin closest to the restoring conveyor when an outgoing unit-load is selected follows P_5.

Figure 3. The main frame of retrieval in Modi-InDeep



PERFORMANCE EVALUATION

In this section, InDeep_n and Modi-InDeep are coded in C++ and executed on a laptop with Intel Core i7-4710MQ and 16GB DDR3L RAM. Let t_h = 1.0 second, t_v = 2.0 seconds, L = 20, H = 10 and M = 10. Let N = 100 and s takes values of 0.065, 0.139, 0.222, 0.317 and 0.431 to make the 20% fast-turnover unit-loads contribute 90%, 80%, 70%, 60% and 50% turnover unit-loads, respectively. The dwell point of the storage machine is the middle point of the storage face and the dwell point of the retrieval machine is the drop-off station.

An instance contains 1,600 warm-up unit-loads, 2,000 storage requests and 2,000 retrieval requests. The references of warm-up, incoming and outgoing unit-loads follow the ABC-demand curve. The storage and retrieval requests are randomly sequenced. For each storage request, empty segments exist. For each retrieval request with form (0, *Ref*(*l*)), there are *Ref*(*l*) unit-loads stored on the flow-rack to meet the requirement of retrieval request. At the beginning of each instance, InDeep_n and Modi-InDeep store warm-up unit-loads to the flow-rack. Then, InDeep_n and Modi-InDeep generate storage and retrieval operations for the storage and retrieval requests.

For each value of s, 100 instances are generated to obtain the average results. Let s_{total} and s_{pure} be the total travel time and travel time without considering restoring of storage machine. Correspondingly, let r_{total} and r_{pure} be the total travel time and travel time without considering removing of retrieval machine. Therefore, $\frac{s_{total}}{2000}$, $\frac{s_{pure}}{2000}$, $\frac{r_{total}}{2000}$ and $\frac{r_{pure}}{2000}$ are employed as the performance measures. ARPD (Average Relative Percentage Deviation) is employed to evaluate the effectiveness of methods.

Table 1 demonstrates the average values of $\frac{s_{total}}{2000}$ (ST) and $\frac{r_{total}}{2000}$ (RT), 95% confident intervals and ARPDs obtained by InDeep_20, InDeep_60, InDeep_100 and Modi-InDeep, respectively. InDeep_100 obtains lowest storage cost and lowest retrieval cost among InDeep_n methods. Therefore, the class separation has significant influence of the performance of storage and retrieval. However, Modi-InDeep method gains shorter storage cost and retrieval cost than InDeep_100 does. Therefore, the modification proposed in this paper is practicable and can improve the performance for the storage assignment in flow-rack AS/RS with class-based storage policy.

Table 1. Travel times of storages and retrievals with different values of *s*

	InDeep_20				InDeep_60				InDeep_100				Modi-InDeep			
s	ST	ARPD	RT	ARPD	ST	ARPD	RT	ARPD	ST	ARPD	RT	ARPD	ST	ARPD	RT	ARPD
.065	31.52	35.16	23.92	120.14	27.44	17.66	19.41	78.54	26.14	12.08	18.48	70.04	23.33	0.00	10.92	0.00
	[31.29, 31.76]		[23.46, 24.37]		[27.24, 27.64]		[18.92, 19.90]		[25.93, 26.36]		[17.98, 18.99]		[23.25, 23.41]		[10.74, 11.10]	
.139	36.01	41.41	30.10	89.65	30.82	21.04	24.83	56.76	27.79	9.13	21.81	36.72	25.47	0.00	16.01	0.00
	[35.74, 36.28]		[29.54, 30.43]		[30.62, 31.02]		[24.39, 25.19]		[27.61, 27.96]		[21.22, 22.07]		[25.34, 25.59]		[15.59, 16.12]	
.222	38.49	42.80	33.09	74.74	33.17	23.05	28.31	49.44	28.76	6.69	23.61	24.55	26.96	0.00	18.97	0.00
	[38.29, 38.69]		[32.79, 33.40]		[32.99, 33.34]		[28.02, 28.59]		[28.63, 28.89]		[23.31, 23.90]		[26.86, 27.06]		[18.77, 19.17]	
.317	39.94	42.27	34.92	65.52	34.80	23.96	30.67	45.32	29.51	5.14	25.24	19.59	28.07	0.00	21.12	0.00
	[39.73, 40.14]		[34.70, 35.14]		[34.64, 34.96]		[30.46, 30.88]		[29.42, 29.61]		[25.07, 25.40]		[27.99, 28.16]		[20.96, 21.28]	
.431	40.72	40.93	35.75	59.79	36.00	24.61	32.28	44.29	30.22	4.61	26.51	18.50	28.89	0.00	22.38	0.00
	[40.50, 40.93]		[35.52, 35.97]		[35.83, 36.18]		[32.10, 32.46]		[30.14, 30.30]		[26.39, 26.63]		[28.82, 28.97]		[22.26, 22.49]	

The box-plots of numbers of blocking unit-loads obtained by InDeep_n and Modi-InDeep methods are illustrated in Figure 4. InDeep_100 returns fewest blocking unit-loads among InDeep_n methods. Consequently, the class separation determines the number of blocking unit-loads. Modi-InDeep method obtains slightly fewer blocking unit-loads than InDeep_100 does. Therefore, travel times for removing and restoring of blocking unit-loads can be saved by InDeep_100 and Modi-InDeep, which lets InDeep_100 and Modi-InDeep obtain lower costs of warehouse operations than InDeep_20 and InDeep_60 do.

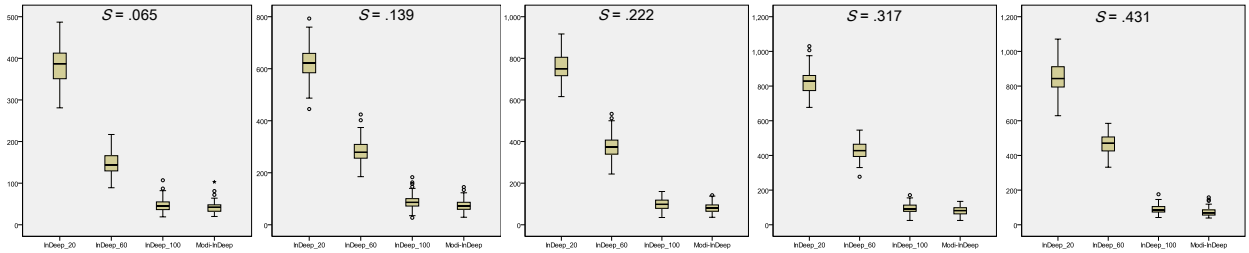


Figure 4. Numbers of blocking unit-loads obtained by InDeep_n and Modi-InDeep

Table 2. Pure travel times of storages and retrievals with different values of *s*

	InDeep_20				InDeep_60				InDeep_100				Modi-InDeep			
s	ST	ARPD	RT	ARPD	ST	ARPD	RT	ARPD	ST	ARPD	RT	ARPD	ST	ARPD	RT	ARPD
.065	26.38	16.08	19.16	85.07	25.47	12.06	17.57	69.66	25.53	12.33	17.97	73.50	22.73	0.00	10.40	0.00
	[26.18, 26.59]		[18.72, 19.60]		[25.28, 25.66]		[17.08, 18.06]		[25.31, 25.75]		[17.45, 18.49]		[22.67, 22.80]		[10.22, 10.57]	
.139	27.61	13.21	22.20	49.20	26.81	9.95	20.98	40.99	26.63	9.21	20.63	38.51	24.40	0.05	14.92	0.03
	[27.41, 27.80]		[21.80, 22.60]		[26.63, 27.00]		[20.58, 21.39]		[26.45, 26.82]		[20.19, 21.07]		[24.29, 24.51]		[14.65, 15.18]	
.222	28.43	10.36	23.90	33.26	27.78	7.84	23.17	29.19	27.48	6.68	22.48	25.31	25.81	0.20	17.96	0.06
	[28.27, 28.59]		[23.59, 24.20]		[27.63, 27.92]		[22.87, 23.47]		[27.34, 27.62]		[22.17, 22.79]		[25.71, 25.91]		[17.75, 18.17]	
.317	28.97	8.49	25.01	24.38	28.58	7.01	24.75	23.08	28.31	6.02	24.17	20.21	26.95	0.92	20.13	0.02
	[28.87, 29.07]		[24.85, 25.17]		[28.49, 28.66]		[24.58, 24.92]		[28.22, 28.41]		[23.99, 24.36]		[26.88, 27.03]		[19.96, 20.29]	
.431	29.34	8.04	25.59	19.15	29.19	7.51	25.82	20.21	29.04	6.93	25.45	18.51	27.88	2.66	21.49	0.04
	[29.26, 29.42]		[25.46, 25.72]		[29.12, 29.27]		[25.69, 25.94]		[28.96, 29.11]		[25.33, 25.57]		[27.81, 27.95]		[21.37, 21.61]	

Table 2 demonstrates average values of $\frac{s_{pure}}{2000}$ (ST), $\frac{r_{pure}}{2000}$ (RT), 95% confident intervals and ARPDs obtained by InDeep_20, InDeep_60, InDeep_100 and Modi-InDeep, respectively. Smaller value of s_{pure} implies that more unit-loads are stored to bins near to the pick-up station. In other words, more unit-loads can be retrieved from bins near to the drop-off station with smaller value of s_{pure}. Table 2 validates that the values of s_{pure} obtained by InDeep_20, InDeep_60 and InDep_100 are similar. As a result, the value of r_{pure} cannot be significantly diminished by changing class separation. The values of s_{pure} and r_{pure} obtained by Modi-InDeep method are shorter than those gained by InDeep_20, InDeep_60 and InDeep_100. Therefore, the modification of the In-deep method can not only reduce the number of blocking unit-loads but also assign more unit-loads to bins near the pick-up station.

CONCLUSIONS

In this paper, the class separation of the In-deep method is analyzed. Experimental results demonstrate that treating all unit-loads as one class can minimize the costs of warehouse operations by reducing number of blocking unit-loads. Based on this, a Modi-InDeep method is

designed, in which all unit-loads are assigned into the same class. In addition, Modi-InDeep method alters some bin selection rules in the storage assignment and outgoing unit-load choosing rules in the retrieval generation. Experimental results validate that Modi-InDeep method has better performance than InDeep_100 does.

The future work should focus on (1) checking the performance of Modi-InDeep method with different numbers of types or different numbers of storage and retrieval requests and (2) employing Modi-InDeep method to similar AS/RSs such as bi-directional flow-rack AS/RS.

ACKNOWLEDGEMENT

This work was supported by National Natural Science Foundation for Young Scientists of China [71602169] and by Natural Science Foundation of the Higher Education Institutions of Jiangsu Province, China [16KJB520045].

REFERENCES

Cardin, O., Castagna P., Sari Z. & Meghelli N. (2012). Performance evaluation of in-deep class storage for flow-rack AS/RS. *International Journal of Production Research*, 50(23), 6775-6791.

Chang T.-H., Fu H.-P., & Hu K.-Y. (2006). Innovative application of an integrated multi-level conveying device to a mobile storage system. *The International Journal of Advanced Manufacturing Technology*, 29(9-10), 962–968.

Chen, Z., Li X., & Gupta J.N.D. (2015). A bi-directional flow-rack automated storage and retrieval system for unit-load warehouses. *International Journal of Production Research*, 53 (14), 4176-4188.

De Koster R.B.M, Le-Duc T., & Yu Y. (2008). Optimal storage rack design for a 3-dimensional compact AS/RS. *International Journal of Production Research*, 46(6), 1495–1514.

Gue K. R., Furmans K., Seibold Z., & Uludag O. (2014). Gridstore: a puzzle-based storage system with decentralized control. *IEEE Transactions on Automation Science and Engineering*, 11(2), 429–438.

Guo X., Yu Y., & De Koster R.B.M. (2016). Impact of required storage space on storage policy performance in a unit-load warehouse. *International Journal of Production Research*, 54(8), 2405-2418.

Lerher T., Sraml M., Potrc I., & Tollazzi T. (2010). Travel time models for double-deep automated storage and retrieval systems. *International Journal of Production Research*, 48(11), 3151–3172.

Sari Z., Saygin C., & Ghouali N. (2005). Travel-time models for flow-rack automated storage and retrieval systems. *The International Journal of Advanced Manufacturing Technology*, 25(9-10), 979–987.

Yu M., & De Koster R.B.M. (2009). The impact of order batching and picking area zoning on order picking system performance. *European Journal of Operational Research*, 198(2), 480-490.

Yu Y., De Koster R.B.M., & Guo X. (2015). Class-based storage with a finite number of items: using more classes is not always better. *Production and Operations Management*, 24(8),1235-1247.

Zaerpour N., Yu Y., & De Koster R.B.M. (2017a). Small is beautiful: a framework for evaluating and optimizing live-cube compact storage systems. *Transportation Science*, 51(1), 23-51.

Zaerpour N., Yu Y., & De Koster R.B.M. (2017b). Response time analysis of a live-cube compact storage system with two storage classes. *IISE Transactions*, 49(5), 461-480.