

國立臺灣大學工學院工業工程學研究所

碩士論文



Graduate Institute of Industrial Engineering

College of Engineering

National Taiwan University

Master's Thesis

物料搬運網路排程問題的數學規劃

及啟發式優化求解法

Mathematical Programming and Metaheuristics for
the Material Handling Network Scheduling Problem

陳徐行

Hsu-Hsing Chen

指導教授：楊烽正 博士

Advisor: Feng-Cheng Yang, Ph.D.

共同指導教授：洪英超 博士

Co-Advisor: Ying-Chao Hung, Ph.D.

中華民國 113 年 6 月

June, 2024



國立臺灣大學碩士學位論文
口試委員會審定書
MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY


論文中文題目：物料搬運網路排程問題的數學規
劃及啟發式優化求解法

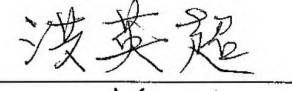
論文英文題目：Mathematical Programming and
Metaheuristics for the Material
Handling Network Scheduling
Problem

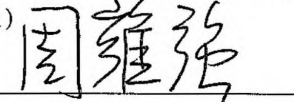
本論文係陳徐行君（學號 R11546019）在國立臺灣大學工業工程學研
究所完成之碩士學位論文，於民國 113 年 6 月 21 日承下列考試委員審
查通過及口試及格，特此證明。

The undersigned, appointed by the Institute of Industrial Engineering on 21 June 2024, have
examined a Master's thesis entitled above presented by CHEN, HSU-HSING (Student ID:
R11546019) candidate and hereby certify that it is worthy of acceptance.

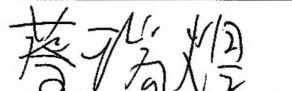
口試委員 Oral examination committee :

楊烽正 
(指導教授)

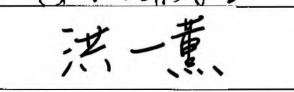
洪英超 
(共同指導教授)

周雍強 

歐陽超 

蔡瑞煌 

所長 Director

洪一薰 



致謝

時光荏苒，一年多的碩士生涯終於要告一個段落。首先要感謝楊老師過去五年來的提攜與照顧；從大二的程式設計課開始，到大三修習了老師在工工所開設的課程，一步步帶領我進入工業工程的領域。嗣後又指導了我進行大專生研究計畫，讓我獲得難能可貴的獨立研究之機會。在研究所的日子，很感謝老師給予了我一定的研究自由，讓我能夠根據我的步調，一步步地完成我的碩士研究，並在其中適時地提供協助。另外，也要感謝老師在我考取公費留學及申請學校的過程中，總是願意提供可貴的幫助。在此敬祝楊老師有一個健康快活的退休生活！

此外，也要特別感謝口試委員 蔡瑞煌老師、歐陽超老師、洪英超老師以及周雍強老師，在我的論文口試中積極地指出不足之處，並提供寶貴的建議。尤其要感謝周老師，願意提攜後進，將相關的專業知識傾囊相授，並且一針見血地點出可改進之處，謝謝老師！

接下來我要感謝我自己。感謝我在這三年來艱苦的學校申請過程中沒有放棄自己，始終相信自己的能力。希望在未來的學術生涯中，我能夠盡興地研究自己有興趣之課題，勇往直前，並對社會有所貢獻。另外，我也要感謝實驗室的夥伴陳禎。這一年多以來的互相砥礪，是我能如期畢業的一大原因。此外，我也要感謝林以達學長，讓我有機會誤打誤撞挑了這個研究題目，並從中學習到了很多相關技術，最終撰寫成論文。

最後，我要感謝陳爸爸和徐媽媽。感謝你們一直以來無私奉獻與支持，我才能夠無後顧之憂地暢遊於學習的樂趣中，並且讓我在學術生涯的起點上有了充足的準備。預祝您們健康快樂、長命百歲！

陳徐行 謹謝

2024年夏 於台大國青



Abstract



Modern factories heavily rely on Automated Material Handling Systems (AHMSs) for internal logistics. This research aims to tackle the scheduling problem in the material handling network. We rigorously define the Material Handling Network Scheduling Problem (MHNSP) with the optimization goal of minimizing the makespan for a set of transportation jobs. The highlights of our research are that each job has path flexibility with multiple candidate paths, and the transfer (conveyor) systems between AHMSs have buffer capacity limits.

We propose three models to solve the MHNSP: a Constraint Programming (CP) model, an Integer Programming (IP) model, and a Metaheuristic model. The CP model employs a hierarchical structure to model the constraints between jobs, paths, and operations, while the IP model directly determines the start time of each operation. The IP model addresses site buffer constraints by identifying operation overlaps using pairwise relationships. Finally, the Metaheuristic model utilizes a discrete-event-based decoding procedure to determine the start time of each operation.

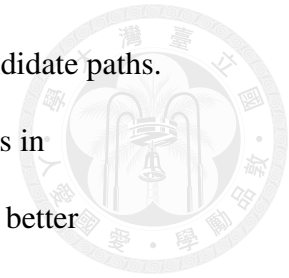
The models are evaluated through four numerical tests. First, we identify the best parameters for the Metaheuristic model using the Taguchi method. Next, we compare the performance of our models using 360 randomly generated numerical test problems. The results reveal distinct strengths: the IP model is effective for small problems, while the CP and Metaheuristic models are better suited for larger problems. Additionally, test results show a significant average makespan reduction

of about 13% in large-scale problems when each job has multiple candidate paths.

We also evaluate the performance of the CP and Metaheuristic models in extra-large problems, finding that the CP model consistently provides better solutions than the Metaheuristic model given sufficient solving time, demonstrating the potential of CP in real applications. Finally, the identical request test highlighted that optimal path selection could lead to a 35.7% reduction in makespan by balancing node workloads and minimizing time spent at transfer sites.

In conclusion, this research underscores the importance of path selection and operation sequencing in optimizing material handling networks, providing robust models and comprehensive evaluations to guide future applications.

Keywords: *Job-shop Scheduling Problem, Material Handling System, Constraint Programming, Integer Programming, Metaheuristics, Path Flexibility*



摘要



現代加工廠高度依賴自動化物料搬運系統 (AHMS) 來進行內部物流。本研究針對物料搬運網路中的排程問題進行探討，並定義了物料搬運網路排程問題 (Material Handling Network Scheduling Problem, MHNSP)。該問題的最佳化目標是最小化給定之搬運工作的總完工時間 (makespan)。本研究的兩個特色，一是每個搬運工作具有多條候選路徑，二是搬運系統之間的輸送系統具有容量限制。

我們提出了三種求解 MHNSP 的模型：限制規劃 (CP) 模型、整數規劃 (IP) 模型和一種啟發式優化模型。其中，限制規劃模型採用多層次結構，以建構搬運工作、候選路徑及搬運作業 (operation) 間的限制式。而整數規劃模型則直接求解各搬運作業的開始時間；另外，整數規劃模型透過識別搬運作業間在時間上重疊的關係，以建構對於輸送系統容量限制的限制式。最後，我們的啟發式優化模型是透夠一個基於離散事件模擬的解碼程序，來找出各個搬運作業的開始時間。

為瞭解本研究問題在實務上的應用，及各個求解法在不同標竿問題的表現，我們通過四個數值測試對這些求解法進行評估。首先，我們使用田口方法找出啟發式優化模型的最佳參數。接下來，我們使用 360 個隨機生成的標竿問題來比較模型的性能。結果顯示，整數規劃模型適用於小型問題，而限制規劃和啟發式優化模型更適合大型問題。此外，測試結果顯示，在大型問題中，當每個搬運工作有多條候選路徑時，平均總完工時間約減少 13%。我們還在超大型問題中測試了限制規劃和啟發式優化模型的表現，結果顯示限制規劃模型在給定充足求解時間的情況下，其求解品質皆優於啟發式優化模型。此測試應證了限制規劃模型在實際應用中的潛力。最後，我們透過

「重複搬運任務測試」(Identical Request Test)，展示了在路徑選擇最佳化的情況下，相對於固定搬運路徑的策略，總完工時間減少了 35.7%。透過對求解結果分析，可知其減少來源有二，一是搬運系統間有較平衡的工作附載，二是在物料在輸送系統中等待的時間大幅減少。

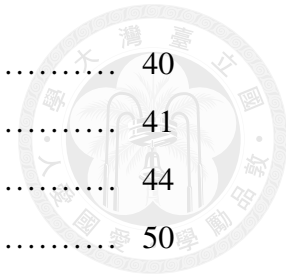
總之來說，本研究強調了物料搬運網路中路徑選擇和搬運作業排序的重要性，並提供了大量的數值測試結果以佐證。

關鍵字：零工生產排程問題、物料搬運系統、限制規劃、整數規劃、啟發式優化模型、彈性路徑。

Table of Contents

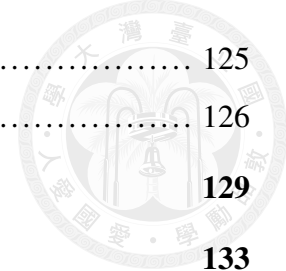


Abstract	v
Table of Contents	ix
List of Figures	xiii
List of Tables	xv
Glossary and Notations	xvii
1 Introduction	1
1.1 Background and Motivation	1
1.2 Research Objectives	3
1.3 Research Procedure	4
1.4 Organization of the Thesis	6
2 Literature Review	7
2.1 Transportation Network Job Scheduling Problem	8
2.2 Importance of Path Flexibility in Transportation	9
2.3 Job-shop Scheduling Problem with Routing Flexibility	10
2.4 Job-shop Scheduling Problem Incorporating Material Handling Sys- tems Using Constraint Programming	12
2.5 CP Overview	13
2.6 Summary	17
3 Material Handling Network Scheduling Problem: Problem Descrip- tion and Problem Generation	19
3.1 Problem Definition	20
3.1.1 Problem Overview	20
3.1.2 Mathematical Formulation	23
3.1.3 Data Structure of a Solution	31
3.1.4 Assumptions and Problem Scope	32
3.1.5 Summary	34
3.2 Numerical Test Problem Generation	39



3.2.1	User-defined Parameters	40
3.2.2	Network Construction.....	41
3.2.3	Transportation Job Generation	44
3.2.4	Problem Scales, Types and File Format.....	50
3.2.5	Summary	57
4	Constraint Programming, Integer Programming, and Differential Evolution Models for Material Handling Network Scheduling Problem	59
4.1	CP Model for solving the MHNSP.....	59
4.1.1	Job Constraints	60
4.1.2	Operation Constraints on Nodes.....	68
4.1.3	Operation Constraints on Transfer Sites	74
4.2	IP Model for solving the MHNSP.....	76
4.2.1	Decision Variables and Optimization Goal	77
4.2.2	Variable Constraints on Each Job	79
4.2.3	Operation Constraints on Nodes.....	80
4.2.4	Transfer Operation Constraints on Transfer Sites.....	83
4.3	Metaheuristic Model for solving the MHNSP	88
4.3.1	Solution Encoding and Decoding	89
4.3.2	Permutational Differential Evolution Solver	102
4.4	Summary	106
5	Numerical Tests and Result Discussion	107
5.1	Solving Method Implementations.....	107
5.2	Numerical Tests and Discussion.....	109
5.2.1	Differential Evolution Model Parameter-tuning Experiment via the Taguchi method	110
5.2.2	Model Performance Comparison Test.....	113
5.2.3	Extra-large Problem Test.....	116
5.2.4	Identical Request Test.....	117
5.3	Summary	121
6	Conclusion and Future Work Suggestion	123
6.1	Conclusion.....	123
6.2	Future Work	125

6.2.1	Improvements in Modeling Techniques	125
6.2.2	Improvements in Research Problems.....	126
Reference		129
A	Different Type of Transportation Nodes	133
A.1	CP Model for Different Types of Transportation Nodes	135
B	Numerical Result Raw Data	141



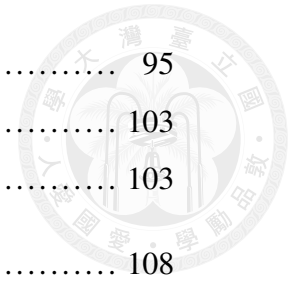




List of Figures

1-1	Illustration of the research motivation.	3
1-2	Flow chart of our research procedure.	5
3-1	Illustration of a material handling network.....	20
3-2	An illustration of multiple candidate paths for a request.	22
3-3	Graph representation of the network example.	24
3-4	Definition of a transportation job.	25
3-5	Illustration of empty-car moving time between delivery operations.	29
3-6	Illustration of operation sequence in jobs.	30
3-7	A small example of the MHNSP.....	35
3-8	Gantt chart of Solution A for the MHNSP sample.....	36
3-9	Gantt chart of Solution B for the MHNSP sample.....	37
3-10	Gantt chart of Solution C for the MHNSP sample.....	37
3-11	Gantt chart of Solution D for the MHNSP sample.....	38
3-12	Comparison of generated networks in different scales.....	51
3-12	Comparison of generated networks in different scales. (cont.)	52
3-13	Directory structure of the numerical test problem files.....	54
4-1	Illustration of the hierarchical structure of the interval variables.	61
4-2	Gantt chart results of a CP solution.	62
4-3	Alignment Relationship between job interval variable, path interval variable, and operation interval variable.....	66
4-4	Gantt chart result of nodes from the CP model.	69
4-5	Illustration of transfer operation interval variables and cumulative function of resource.	75
4-6	Gantt chart results of an IP solution.	77
4-7	Gantt chart results of an IP solution on nodes.	81
4-8	Illustration of two examples of three transfer operations in the site.....	84
4-9	Illustration of two examples of three transfer operations in the site.....	91
4-10	Illustration of the concept in the first stage of the decoding procedure.	93

4-11 A material handling network for deadlock demonstration.....	95
4-12 An example of order-base crossover.	103
4-13 An example of swap mutation.	103
5-1 Directory structure of the implemented solvers.....	108
5-2 Screenshot of the source code of solver implementations on GitHub.	109
5-3 Node Gantt charts yielded from the fixed and flexible problems.....	119
5-4 Job Gantt charts yielded from the fixed and flexible problems.....	120



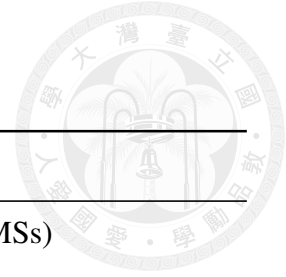


List of Tables

3-1	Summary of user-defined parameters for problem generation.	41
3-2	Parameters for problem generation of different scales.	50
3-3	Comparison of different problem types.	54
5-1	Design level for each DE parameter.	110
5-2	Taguchi L9 design.	111
5-3	Taguchi L9 design parameter selection.	112
5-4	Best parameters of each problem setting.	112
5-5	Hit rate and average solving time of model performance comparison test.	114
5-6	Makespan improvement of the flexible model over the fixed model.	116
5-7	Makespan comparison between CP and DE solvers in Extra-large Problems.	117
5-8	Numerical results for the identical request test.	118
5-9	Reorganized numerical results for the identical request test.	119
B-1	Taguchi results of DE parameter-tuning.	141
B-2	Objective values of model performance comparison test.	144
B-3	Solving time of model performance comparison test.	147

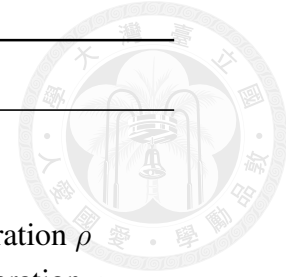


Glossary and Notations



Category	Notation	Description
Material Handling Network	m	Number of transportation nodes (AHMSs)
	r	Number of transfer sites
	V	Index set of transportation nodes; $V = \{1, 2, \dots, m\}$
	e_j	Transfer site e_j
	E	Edge set of transfer sites; $E = \{e_1, e_2, \dots, e_r\}$
	G	Graph of material handling network; $G = (V, E)$
	κ_i	Number of P/D points in node i
	$P^{(i)}$	P/D point set of node i ; $P^{(i)} = \{p_1^{(i)}, p_2^{(i)}, \dots, p_{\kappa_i}^{(i)}\}$
	$\bar{\kappa}$	Upper bound of number of P/D points in a node (problem generation)
	$\tau_{j,j'}^{(i)}$	Delivery time from P/D point $p_j^{(i)}$ to $p_{j'}^{(i)}$ in node i
	$M^{(i)}$	From-to time matrix of node i ; $M^{(i)} = [\tau_{j,j'}^{(i)}]_{\kappa_i \times \kappa_i}$
	$\bar{\tau}$	Maximum delivery time in a node (problem generation)
	β_d	Transfer time in transfer site e_d
	$\bar{\beta}$	Upper bound of transfer time (problem generation)
	ω_d	Site capacity of transfer site e_d
$\bar{\omega}$	Upper bound of site capacity (problem generation)	
Transportation Job	C^{\max}	Makespan
	n	Number of transportation requests/jobs
	J_k	Transportation request/job J_k ; $J_k \equiv (g_k, p_{\zeta_k}^{(\xi_k)}, p_{\zeta'_k}^{(\xi'_k)}, \Pi_k)$
	J	Set of transportation requests/jobs; $J = \{J_1, J_2, \dots, J_n\}$
	g_k	Request generated time of job J_k
	\bar{g}	Upper bound of request generated time (problem generation)
	$p_{\zeta_k}^{(\xi_k)}$	Start P/D point of job J_k ; $p_{\zeta_k}^{(\xi_k)} \in P^{(\xi_k)}$
	$p_{\zeta'_k}^{(\xi'_k)}$	End P/D point of job J_k ; $p_{\zeta'_k}^{(\xi'_k)} \in P^{(\xi'_k)}$
	σ_k	Number of candidate paths in job J_k
	$\pi_l^{(k)}$	The l -th candidate path of job J_k ; $\pi_l^{(k)} = \langle \rho_{l,1}^{(k)}, \rho_{l,2}^{(k)}, \dots, \rho_{l,\lambda_{k,l}}^{(k)} \rangle$
	$\lambda_{k,l}$	Number of delivery operations in the l -th candidate path of job J_k
	$\Pi^{(k)}$	Candidate path set of job J_k ; $\Pi^{(k)} = \{\pi_1^{(k)}, \pi_2^{(k)}, \dots, \pi_{\sigma_k}^{(k)}\}$

Continued on next page

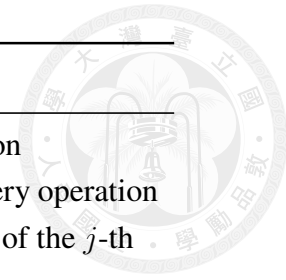


Category	Notation	Description
Constraint Programming	ρ	Delivery operation $\rho \equiv (\eta, \theta, \delta)$
	η	Node index of delivery operation ρ
	θ	Pick-up P/D point index of the delivery operation ρ
	δ	Drop-off P/D point index of the delivery operation ρ
	$\epsilon_{l,s}^{(k)}$	The transfer site that executes the s -th transfer operation in the l -th candidate path of job J_k
	${}^s t_k$	Start time of job J_k
	${}^c t_k$	Completion time of job J_k
	s	Start time of an interval variable; $s \in \mathbb{Z}$
	e	End time of an interval variable; $e \in \mathbb{Z}$
	\perp	Absence of an interval variable in the solution
	T_k	Interval variable of job J_k
	\mathbf{T}	Set of interval variables of all jobs; $\mathbf{T} = \{T_1, T_2, \dots, T_n\}$
	$t_l^{(k)}$	Interval variable of the l -th candidate path of job J_k
	C_k	Set of path interval variables of job J_k ; $C_k = \{t_1^{(k)}, t_2^{(k)}, \dots, t_{\sigma_k}^{(k)}\}$
	$o_{l,s}^{(k)}$	Interval variable of the s -th delivery operation in the l -th candidate path of job J_k
	$O_l^{(k)}$	Set of interval variables of all delivery operations in the l -th candidate path of job J_k ; $O_l^{(k)} = \{o_{l,1}^{(k)}, o_{l,2}^{(k)}, \dots, o_{l,\lambda_{k,l}}^{(k)}\}$
	$\tilde{o}_{l,s}^{(k)}$	Interval variable of the s -th transfer operation in the l -th candidate path of job J_k
	$\tilde{O}_l^{(k)}$	Set of interval variables of all transfer operations in the l -th candidate path of job J_k ; $\tilde{O}_l^{(k)} = \{\tilde{o}_{l,1}^{(k)}, \tilde{o}_{l,2}^{(k)}, \dots, \tilde{o}_{l,\lambda_{k,l}-1}^{(k)}\}$
	$o_0^{(i)}$	Dummy interval variable of the node i
	$X^{(i)}$	Set of interval variables of all delivery operations in node i
$Y^{(i)}$	List of interval variable types of all delivery operations in node i	
$\phi_{j,j'}^{(i)}$	Transition time (empty-car moving time) from interval variable type j to j' in node i	
$\Phi^{(i)}$	Transition time matrix of any pair of interval variable type on node i ; $\Phi^{(i)} = [\phi_{j,j'}^{(i)}]_{\kappa_i^2 \times \kappa_i^2}$	

Continued on next page

Category	Notation	Description
Integer Programming	$\Gamma^{(d)}$	Set of interval variables of all transfer operations in transfer site e_d
	$x_{l,s}^{(k)}$	Start time of the s -th delivery operation in the l -th candidate path of job J_k
	$\tilde{x}_{l,s}^{(k)}$	Start time of the s -th transfer operation in the l -th candidate path of job J_k
	$z_l^{(k)}$	Binary variable of the l -th candidate path of job J_k
	γ_i	Number of delivery operations in node i
	$X^{(i)}$	Set of start time of all delivery operations in node i
	$Y^{(i)}$	Set of operation details of all delivery operations in node i
	$q_{j',j}^{(i)}$	Binary variable of preceding relationship between operations j' to j in node i
	$\bar{q}^{(i)}$	Binary variable of whether the number of present delivery operations in node i is greater than 0
	L	Sufficiently large number
	ν	Sufficiently small number
	$\tilde{\gamma}_i$	Number of transfer operations in site e_d
	$\tilde{X}^{(d)}$	Set of start time of all transfer operations in transfer site e_d
	$\tilde{Y}^{(d)}$	Set of operation details of all transfer operations in transfer site e_d
	$\hat{Y}^{(d)}$	Set of end time of all transfer operations in transfer site e_d
	$a_{j',j}^{(d)}$	Binary variable of whether operation j' overlaps with j in time in transfer site e_d
	$\hat{a}^{(d)}$	Binary variable of whether the start time of operation j' is the same as that of j in time in transfer site e_d
	$\bar{\omega}^{(d)}$	Upper limit of the number of pair-wise overlapping operations in site d ; $\bar{\omega}^{(d)} = \frac{\omega_d(\omega_d+1)}{2}$
	$A_j^{(d)}$	Index set for identifying pair-wise overlapping relationships; $A_j^{(d)}, j = 1, 2, \dots, \tilde{b}_d$
	\tilde{b}_d	Number of index sets for identifying pair-wise overlapping relationships; $\tilde{b}_d = \binom{\tilde{\gamma}_d}{\omega_d+1}$
	$\bar{\lambda}$	Total number of delivery operations decomposed from all candidate paths of all jobs.
	W	Indexed operation set; $W = \left\{ \hat{w}_j \equiv \left(\hat{k}_j, \hat{l}_j, \hat{s}_j, \hat{\eta}_j, \hat{\theta}_j, \hat{\delta}_j \right) \mid j = 1, 2, \dots, \bar{\lambda} \right\}$

Continued on next page



Category	Notation	Description
Metaheuristics	\hat{k}_j	Index of the job of the j -th delivery operation
	\hat{l}_j	Index of the candidate path of the j -th delivery operation
	\hat{s}_j	Index of the operation in the candidate path of the j -th delivery operation
	$\hat{\eta}_j$	Node index of the j -th delivery operation
	$\hat{\theta}_j$	Pick-up P/D point index of the j -th delivery operation
	$\hat{\delta}_j$	Drop-off P/D point index of the j -th delivery operation
	\mathbf{w}	Permuted index array (solution encoding); $\mathbf{w} = [w_1 w_2 \cdots w_{\bar{\lambda}}]$, $w_j \in \{1, 2, \dots, \bar{\lambda}\}$, $w_j \neq w_{j'}, \forall j, j' \in \{1, 2, \dots, \bar{\lambda}\}$
	l_k^*	Index of the selected path for job J_k
	$x_{l,s}^{(k)}$	Start time of the s -th delivery operation in the l -th candidate path of job J_k
	$\tilde{x}_{l,s}^{(k)}$	Start time of the s -th transfer operation in the l -th candidate path of job J_k
	$\mathbf{b}_j^{(i)}$	3-tuple representing the j -th execution for node i .
	$B^{(i)}$	Execution sequence of node i .
	$\nu^{(i)}$	Available time of node i .
	ν_k	Available time of job J_k .
	F	Future event list in discrete-event simulation.
	f_j	Event in the future event list F . $f \equiv (\hat{t}, \hat{y}, \hat{i}, \hat{j})$
	\hat{t}	Event time in the 4-tuple event definition.
	\hat{y}	Event type in the 4-tuple event definition (1: start, 2: end).
	\hat{i}	Node index in the 4-tuple event definition.
	\hat{j}	Execution index of operation in the 4-tuple event definition.
	$z^{(i)}$	Execution sequence index of node i .
	z_k	Operation execution sequence index of job J_k .
	b_d	Blocked flag for the transfer site.
\hat{u}_d	Vacancies on transfer site d .	
Problem Type	$1/\infty$	Fixed/Infinite problem type
	$1/n$	Fixed/Finite problem type
	$1'/n$	Fixed-random/Finite problem type
	n/∞	Flexible/Infinite problem type
	n/n	Flexible/Finite problem type



Chapter 1

Introduction

In modern manufacturing, Automated Material Handling Systems (AHMSs) are widely used across the industry. In addition, with the rise of large-scale factories, internal logistics have become increasingly complex, requiring material transportation beyond adjacent process machines. In many cases, long-distance material transportation requires the use of a *material handling network*, where multiple AHMSs work sequentially to complete a transportation job.

When multiple material loads need to be transported simultaneously, and transportation resources in the network are limited, scheduling issues arise. Therefore, this research focuses on the scheduling problem within the material handling network, aiming to improve the completion time (makespan) for a set of transportation jobs.

1.1 Background and Motivation

In the industry, AMHSs are usually controlled and coordinated by the Material Execution System (MES). Unfortunately, the MES usually adopts a real-time job dispatching system for material transportation, where no scheduling is done in advance. For example, an AMHS would only start moving to pick up a material load from the last stopping position when the load arrives at its I/O port. However, this no-value-added waiting time is, in fact, unnecessary since the

moving and arrival times of most AMHSs are stable and predictable, unlike traditional manual transportation.



Previous research has aimed to improve the efficiency of material handling networks by scheduling operations in AMHSs. However, some crucial factors were overlooked. For instance, the limited buffer capacity of conveyor (transfer) systems was often ignored. Additionally, while multiple paths with the same start and end points usually exist in a network, each transportation job was typically assigned a fixed path in the literature, neglecting the potential flexibility.

To address these gaps, we will propose the **Material Handling Network Scheduling Problem** (MHNSP), which incorporates the overlooked factors. In our research, a transportation job can be fulfilled by any of the *candidate paths*, and we account for buffer capacity within the network. Figure 1-1 illustrates the motivation behind our research. Our goal is to determine how the makespan can be improved when the system has greater flexibility in path selection for jobs and operation sequencing in AMHSs. These flexibilities should enhance the overall production efficiency of the material handling network in terms of the makespan. Note that this enhancement may cause an increase in the cost, but we do not consider this aspect as the MHNSP is a single-objective optimization problem to minimize the makespan.

In the literature, related research problems have been mostly solved by Integer Programming, Metaheuristics, Constraint Programming, etc. Therefore, we will follow their approaches to propose a **Constraint Programming model**, an **Integer Programming model**, and a **Metaheuristics model** to solve the MHNSP.

Each method is based on a distinct modeling technique, providing a more credible basis for comparing the solutions yielded by each model and ensuring the optimality of the results. Additionally, by evaluating these models on various generated problems, we can identify the strengths and weaknesses of each approach in different problem types. Finally, we can understand how and how much the makespan improves by comparing the yielded results in different problem types.

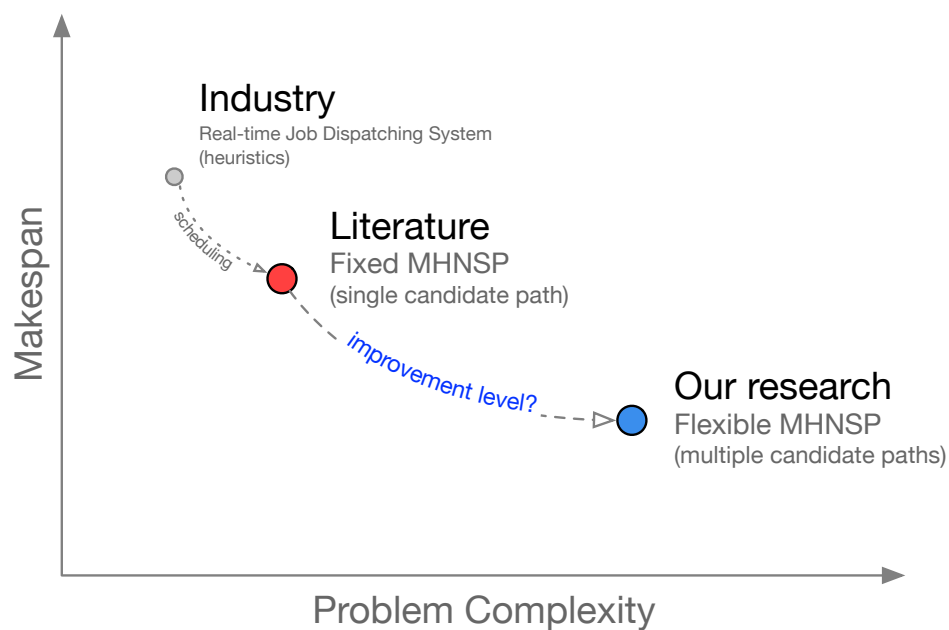
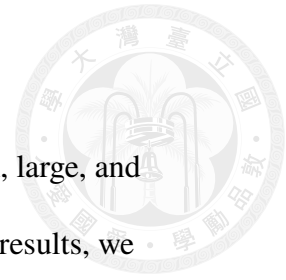


Figure 1-1. Illustration of the research motivation.

1.2 Research Objectives

This research intends to solve the MHNSP using Constraint Programming (CP), Integer Programming (IP), and Metaheuristic algorithm (Differential Evolution). The objectives of our research are:

- Study different modeling techniques to solve the MHNSP. Some key techniques include the hierarchical structure modeling technique in CP, buffer capacity modeling in IP, and the discrete-even-based decoding



procedure in the Metaheuristic algorithm.

- Compare the three proposed solving methods in small, medium, large, and extra-large-scale problems with 5 problem types. Based on the results, we can provide a guideline for choosing the suitable method for different problem settings.
- Understand how the makespan can be improved when considering more candidate paths for each transportation job.

1.3 Research Procedure

The procedure of our research is outlined in the flow chart in Fig. 1-2. As illustrated, we need to iteratively refine and redefine the MHNSP so it can be solved using mathematical programming models while maintaining its practical applicability. In addition, since the three models involve distinct modeling techniques, we must compare the results to ensure the correctness of each model and the feasibility/optimality of the yielded solutions. Finally, since there are thousands of generated problems involved in our numerical tests, we need to organize the numerical results and convert them into meaningful statistics. This process requires data analytics techniques to effectively interpret and present the data.

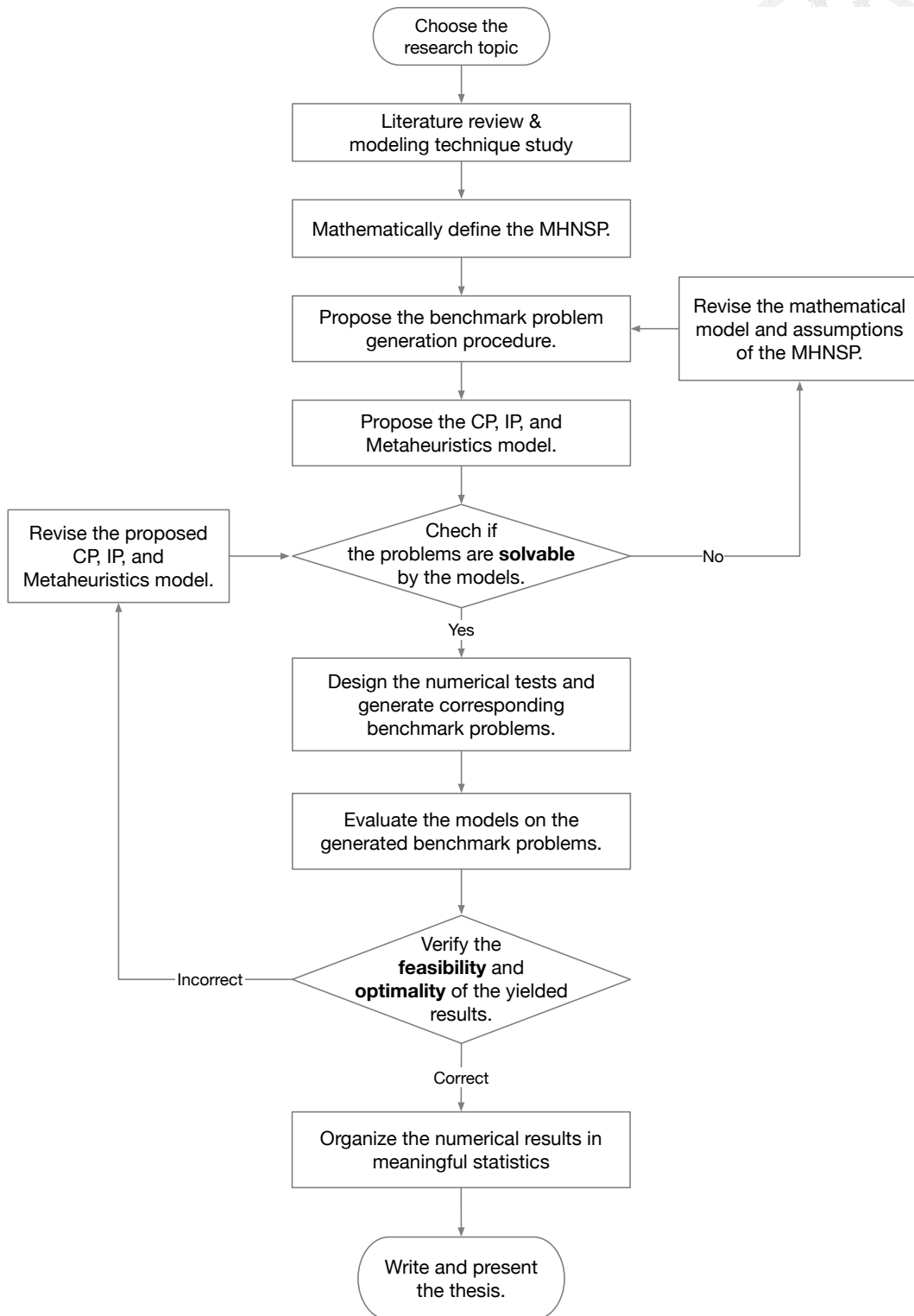


Figure 1-2. Flow chart of our research procedure.

1.4 Organization of the Thesis



We have discussed our research problem, objectives, procedure, and contributions in Chapter 1. In Chapter 2, we will introduce related research and provide a review of Constraint Programming. Next, in Chapter 3, we will formally define the MHNSP and describe the problem generation procedure for the numerical tests.

In Chapter 4, we will detail our solving methods. For the CP and IP models, sections will follow a similar structure, starting with makespan calculation and then deriving constraints for the jobs, nodes (AHMSs), and sites (transfer facilities). For the Metaheuristics, we will focus on the encoding scheme and decoding procedure of the model, followed by an introduction to the metaheuristic algorithm used in our research, the Permutational Differential Evolution.

The numerical test results and discussion will be presented in Chapter 5. Finally, we will conclude our research and suggest future topics in Chapter 6.



Chapter 2

Literature Review

This chapter reviews related research on the material handling network scheduling problem (MHNSP) in the literature.

First, we review the pioneering research that studied the scheduling problem in the material handling network. However, the main drawback was that they did not account for path flexibility for each job. We will review some related research in the transportation field to indicate the importance of path flexibility. Beyond the transportation field, routing flexibility in manufacturing has also been extensively studied. In particular, the Job-shop Scheduling Problem (JSP) variant with routing flexibility has a strong connection with our research problem in terms of model structure. Therefore, we will also review this JSP variant and demonstrate its analogy to our MHNSP.

Another JSP variant that considers the material handling systems in scheduling has also been proposed. In their research, Constraint Programming (CP) has been shown to have great capability in solving large, complex scheduling problems. Consequently, we will also adopt CP to solve the MHNSP. However, since Constraint Programming is relatively unknown to most researchers, we will give an overview of CP in scheduling in the last section of this chapter.



2.1 Transportation Network Job Scheduling Problem

The scheduling problem within the material handling network was first introduced by Lin and Yang [15]. They formally defined the Fixed Transportation Network Job Scheduling Problem (TNSP) and addressed it using a mixed-integer linear programming (MILP) model, five heuristic algorithms, and a genetic algorithm [7]. The TNSP aimed to minimize the makespan, with each transportation job following a fixed routing path. However, their model did not account for buffer capacity and transfer time at transfer sites (conveyor systems) between Automated Material Handling Systems (AHMSs). Their findings indicated that the genetic algorithm provided the best solutions in terms of makespan when optimal solutions were unavailable. Even when the MILP model could solve the problem, the genetic algorithm's results had the smallest gap to the global minimum compared to other heuristic methods. Compared to our research, the TNSP is a specific case of the Material Handling Network Scheduling Problem (MHNSP). As will be discussed in Section 3.2.4, the TNSP corresponds to the $1/\infty$ problem type in our notation, where transfer time is set to zero, and buffer capacity limits are ignored.

The main drawback of Lin and Yang's work is their lack of consideration for multiple candidate paths for each job (path flexibility). Studies have demonstrated that path/routing flexibility can significantly enhance system efficiency and reliability.

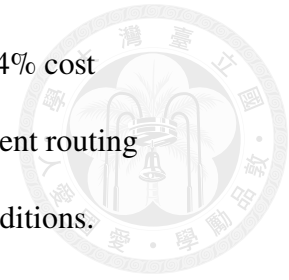
2.2 Importance of Path Flexibility in Transportation

This section reviews three recent works in the transportation field that incorporated path flexibility. All three research suggested that flexibility in path selection could enhance the efficiency of the systems.

Groß et al. [5] addressed the problem of reliable and cost-efficient routing in city logistics. They contended that traditional vehicle routing approaches using precomputed shortest paths based on average travel times failed to account for the variations and uncertainties in travel times due to fluctuating traffic volumes and limited infrastructure. Therefore, they adopted the k-Shortest-Paths (KSP) algorithm to compute a set of alternative routes between customer locations and used the min-max regret approach [11] to select the most reliable candidate path. The result showed that this approach yielded a slightly higher total travel time than the standard average (shortest path) model but achieved much fewer time window violations.

Huang et al. [9] studied the time-dependent vehicle routing problem (TDVRP) by introducing the concept of path flexibility (PF). Traditional vehicle routing problems typically did not account for the varying traffic conditions and the multiple possible paths between customer locations, which can significantly impact travel time and fuel consumption. The main research question was how to integrate path selection in the routing decision-making process under both deterministic and stochastic traffic conditions to minimize overall operational costs. They proposed an MILP model for the TDVRP-PF. The TDVRP-PF model (MILP) achieved considerable savings in operational costs and fuel consumption compared to the

traditional TDVRP model. The deterministic TDVRP-PF showed 2.14% cost savings and 6.37% fuel savings. Path flexibility allows for more efficient routing structures, particularly in scenarios with high variability in traffic conditions.



Guo et al. [6] tackled the challenge of designing efficient urban customized bus routes. A major issue in current systems was unpunctuality caused by traffic congestion. To solve this, the authors proposed a methodology for time-dependent bus route planning that incorporates path flexibility, allowing for different routes between nodes depending on traffic conditions and demand patterns. They developed an MILP model and a hybrid heuristic that incorporates the tabu search and a variable neighborhood search (VNS) procedure. These models integrated bus route planning, path selection, and passenger assignment to minimize operating costs, travel costs, route duration costs, and penalties for delays. Numerical experiments showed that the proposed models effectively integrated path flexibility, resulting in a significant reduction of up to 15.2 % in cost, 26.5% in travel time, and 17.5% in travel distance. In their conclusion, incorporating path flexibility allowed for more adaptable and efficient route planning. This flexibility led to better utilization of bus capacity, reduced travel times, and minimized delays caused by traffic congestion.

2.3 Job-shop Scheduling Problem with Routing Flexibility

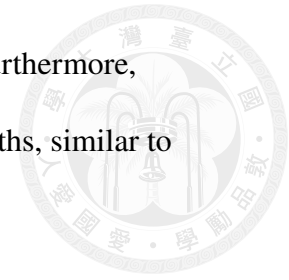
Flexibility in path selection has also been widely studied in research problems in manufacturing. As indicated by Lin and Yang [15], TNSP (or

MHNSP) has a strong relationship with the Job-shop Scheduling Problem (JSP) [21]. In a classic JSP, n jobs must be processed on m machines. Each job involves a specific sequence of operations, each assigned to a particular machine with a defined processing time. It is assumed that each machine can process only one job at a time.

Several variants of JSP have been proposed and extensively studied [26]. One closely related variant to our MHNSP is the JSP with sequence-dependent setup time, sequence-independent setup time, and routing flexibility [2, 3, 8, 10, 18, 19, 20, 24] for flexible manufacturing systems (FMS). In this variant, each job has multiple process plans (routes) to choose from, and there are setup times between consecutive operations on a machine (sequence-dependent setup time) and between consecutive operations in a job's process plan (sequence-independent setup time). This type of JSP variant has been solved by ant colony optimization (ACO) [20], mixed-integer programming (MIP) [2], genetic algorithm (GA) [3], etc. As concluded by Tsubone and Horikawa [24], routing flexibility showed a greater reduction in average flow time (AFT) in random job shop environments and provided superior performance in environments with frequent and long machine breakdowns.

In our MHNSP, each AHMS can be considered a “machine,” and material transportation within an AHMS can be viewed as an “operation.” Due to physical limitations, an AHMS must move from the drop-off point of one operation to the pick-up point of the next operation, analogous to the sequence-dependent setup time. Additionally, each material transfer between AHMSs in the MHNSP requires

a transfer time equivalent to the sequence-independent setup time. Furthermore, each transportation job must choose a path from a set of candidate paths, similar to the routing flexibility in the JSP variant.



The main difference between our research and the JSP variant is our focus on the physical movement of the AHMSs. Consequently, the “process time” and “sequence-dependent setup time” in the MHNSP are derived from the timetable of the AHMS between each pair of pick-up and drop-off points. Additionally, we consider the buffer capacity in transfer facilities, which the JSP variant does not.

2.4 Job-shop Scheduling Problem Incorporating Material Handling Systems Using Constraint Programming

As discussed in the previous section, multiple methodologies have been applied to solve the JSP variant. This section reviews two research that used Constraint Programming in solving another JSP variant incorporating material handling systems in scheduling.

Liu and Yang [16] proposed the Flexible Job and Material Delivery Scheduling Problem. Unlike classic JSP, the material transportation between each machine in their problem was handled by a set of automated guided vehicles (AGVs); therefore, there was a sequence-dependent setup time between operations on each machine incurred by the AGV. Besides, each operation could choose to be processed by one of the machines of the same type, similar to the flexible job-shop scheduling problem. In some sense, this also provided the routing flexibility for

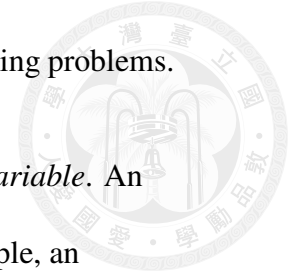
each job. This problem was solved by Constraint Programming, which showed the practical applicability of CP in complex scheduling problems.

Khayat et al. [4] also studied a similar problem of scheduling production tasks and material handling tasks in a job shop environment. However, in their problem, each process machine had a sufficiently large input/output buffer and thus could process the next workpiece immediately without waiting for the material handling vehicle to arrive. They proposed an MILP model and a CP model. The results showed that the MILP model successfully solved test problems with optimal solutions obtained for most small cases. The results also demonstrated that decreasing material handling times and adding more vehicles can significantly increase the solving time. In addition, the CP model showed better performance for larger problem instances, solving them in a few seconds to a few minutes. The model proved to be more efficient than the MILP model for complex job shop environments.

As supported by the above research, CP has great potential in solving complex scheduling problems. Therefore, we will also adopt the CP to solve the MHNSP. In the next section, we will provide an overview of Constraint Programming in scheduling problems.

2.5 Overview for Constraint Programming in Scheduling Problems

Constraint Programming (CP) has been found suitable for many scheduling problems, resource allocating problems, etc. [4, 12, 16, 25] In this section, we will



provide a brief overview of the modeling techniques for CP in scheduling problems.

The cornerstone of CP in scheduling problems is the *interval variable*. An interval variable x occupies a time period on the time axis. For example, an interval variable might be used to present the time span of an operation. However, in many “flexible” problems, an operation might not be processed if not selected. Hence, CP provides a feature to specify an interval variable as “optional”, and let the solver dictate whether the variable participates in the solution or not. The interval variable x can be formally defined by

$$x \in \{[s, e) \mid s, e \in \mathbb{Z}, s \leq e\} \cup \{\perp\}.$$

If $x = \perp$, x would not appear in the solution. The properties of an interval variable x can be extracted via three named operators: $startOf(x)$ returns the start time s of x , $endOf(x)$ returns the end time e of x , $presenceOf(x)$ returns false if the CP solver discards the optional variable x . Formally speaking, we have

$$startOf(x) = s$$

$$endOf(x) = e,$$

where $x \equiv [s, e)$. If $x = \perp$, $startOf(x)$ and $endOf(x)$ are not defined. We also have

$$\begin{cases} presenceOf(x) = false, & \text{if } x = \perp \\ presenceOf(x) = true, & \text{otherwise} \end{cases}$$

For convenience, we use the abbreviations for the three operations:

$$startOf(x) = s(x), \quad endOf(x) = e(x), \quad presenceOf(x) = p(x).$$

There is another function $lengthOf(x) = e(x) - s(x)$ that returns the time span of the interval variable x when present.

Sometimes, we need to define the start/end time relationship between two operations (interval variables). For example, the operation sequence in a JSP must be executed in the given order. For two consecutive operations (interval variables) o_j and o_{j+1} of a job, we have the constraint

$$\text{endBeforeStart}(o_j, o_{j+1}).$$

This constraint mandates that $e(o_j) \leq s(o_{j+1})$; therefore, operation o_j would be executed prior to operation o_{j+1} . There is a set of similar constraints that define the relationship between two interval variables a and b , which are

$$\text{startAtStart}(a, b) \iff \{(p(a) \wedge p(b)) \implies (s(a) = s(b))\}$$

$$\text{startAtEnd}(a, b) \iff \{(p(a) \wedge p(b)) \implies (s(a) = e(b))\}$$

$$\text{endAtStart}(a, b) \iff \{(p(a) \wedge p(b)) \implies (e(a) = s(b))\}$$

$$\text{endAtEnd}(a, b) \iff \{(p(a) \wedge p(b)) \implies (e(a) = e(b))\}$$

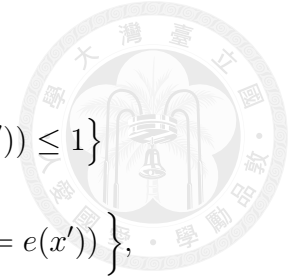
$$\text{startBeforeStart}(a, b) \iff \{(p(a) \wedge p(b)) \implies (s(a) \leq s(b))\}$$

$$\text{startBeforeEnd}(a, b) \iff \{(p(a) \wedge p(b)) \implies (s(a) \leq e(b))\}$$

$$\text{endBeforeStart}(a, b) \iff \{(p(a) \wedge p(b)) \implies (e(a) \leq s(b))\}$$

$$\text{endBeforeEnd}(a, b) \iff \{(p(a) \wedge p(b)) \implies (e(a) \leq e(b))\}.$$

In many flexible scheduling problems, we need to choose exactly one operation from a set of alternatives. In CP modeling, selecting one optional interval from a candidate set for a target interval can be achieved via the constraining operator $\text{alternative}(x, X)$, where X is the candidate interval set for the target interval x . In executing this constraint, the CP solver selects one interval x' from X for x , setting properties of x to x' , i.e., start/end time, and discards other



intervals in X . These restrictions can be defined by

$$\text{alternative}(x, X) \iff \left\{ \left\{ p(x) \iff \bigvee_{x' \in X} p(x') \right\} \wedge \left\{ \text{count}(p(x')) \leq 1 \right\} \right. \\ \left. \bigwedge_{x' \in X} p(x') \implies (s(x) = s(x') \wedge e(x) = e(x')) \right\},$$

where \vee and \wedge are the *or* and *and* operators, respectively. The first two parts state that if x is present, there should be exactly one of the interval variables in X present, and all the others are absent. If x is absent, all the interval variables in X should be absent. The last part states that if $x' \in X$ is present, then the start/end time of x and those of x' should align.

In CP modeling, the constraining operator $\text{span}(x, X)$ sets interval x to represent the covering range of a set of subintervals. Therefore,

$\text{startOf}(x) = \min_{x' \in X} \{\text{startOf}(x')\}$ and $\text{endOf}(x) = \max_{x' \in X} \{\text{endOf}(x')\}$. Moreover, if x is absent, all the interval variables in X should be absent too, namely

$$\text{presenceOf}(x) \iff \bigvee_{x' \in X} \text{presenceOf}(x').$$

There is another variable type called the *sequence variable* in CP that represents an ensemble of interval variables that have temporal relationships. For a given set of interval variables, the CP solver can set permutation orders on their start times by defining a “sequence” variable for a set of intervals to set their ordering sequence. Let $Q \equiv \text{seq}(X)$ be a sequence variable of the CP model to arrange the orders of the interval variables in X . For example, if $X = \{x_1, x_2, x_3\}$, $x_2 \mapsto x_3 \mapsto x_1$ is a value for the sequence variable Q , where $s(x_2) \leq s(x_3) \leq s(x_1)$ yields in the solution.

Finally, in CP modeling, we can also model the resource utility level. The

cumulative function $pulse(x, h)$ defines a pulse covering the range of the interval variable x with the height h . This can be used to represent the usage level of the resource, which is h , by the operation x when it is being processed. By aggregating the cumulative functions defined by a set of interval variables X , that is

$$C = \sum_{x' \in X} pulse(x', h),$$

we can derive the resource utility level profile in time C incurred by these interval variables. If the resource utility level has a minimum or maximum limitation, we can use the cumulative constraint $alwaysIn(C, x, \underline{u}, \bar{u})$ such that the (aggregated) cumulative function C values are within the bounded range $[\underline{u}, \bar{u}]$ in the range of interval variable x . This will be used to model the buffer capacity in the transfer sites.

2.6 Summary

This chapter provided a literature review for the MHNSP. We introduced the TNSP, which was the key reference that inspired our research. We also reviewed some JSP variants that were closely related to the MHNSP. Finally, we provided an overview of the CP in scheduling problems. We focused on introducing the constraints in CP that will be used in the following chapters.

In the next chapter, we will formally define the MHNSP in a mathematical model. Then, we will propose the procedure for numerical test problem generation.





Chapter 3

Material Handling Network Scheduling Problem: Problem Description and Problem Generation

In modern manufacturing systems, smooth and speedy material movement is crucial for overall production efficiency. In particular, when the material control system consists of multiple heterogeneous Automatic Material Handling Systems (AHMSs), the material movements are usually handled by various resources and require proper coordination between the systems. However, effectively coordinating the resources to complete all of the movements is a tough task. As a result, this necessity has given rise to the Material Handling Network Scheduling Problem (MHNSP), a challenge that focuses on optimizing material transportation resources in these systems. The MHNSP is concerned with finding an effective scheduling solution to fulfill jobs as soon as possible, subject to the limited availability of transportation resources such as transport equipment and buffer sizes. The goal is to ensure that all material transportation requests are fulfilled optimally in terms of the completion time.

This chapter will present a rigorous definition of the MHNSP. We will start with constructing a mathematical model for the problem, which will be detailed in Section 3.1.2. Additionally, we will depict the procedures of numerical test problem generation, which are essential for evaluating and refining the proposed solvers.



3.1 Problem Definition

The proposed Material Handling Network Scheduling Problem (MHNSP) poses a challenge in optimizing material handling resources in a complex material handling network to complete a set of transportation jobs. The objective is to minimize the maximum completion time, namely the *makespan* of all jobs. This section rigorously defines the MHNSP, starting with an overview of the problem and progressing toward a more intricate mathematical formulation.

3.1.1 Problem Overview

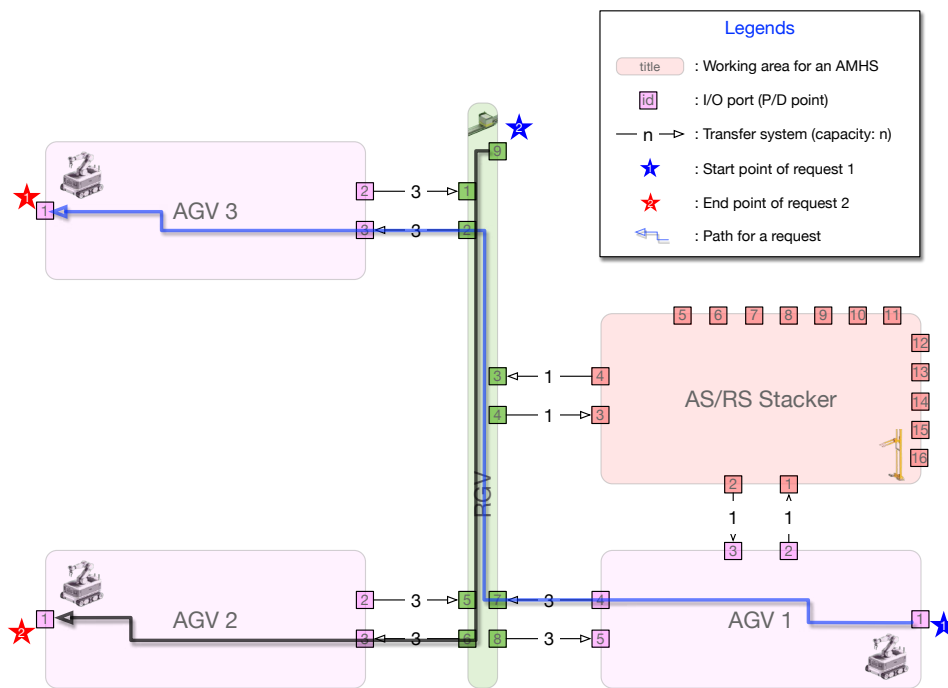


Figure 3-1. Illustration of a material handling network.

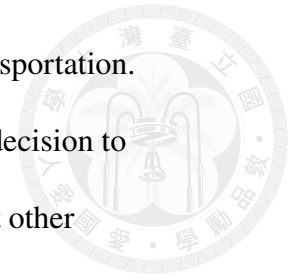
Figure 3-1 depicts a sample material handling network. This network consists of three automated guided vehicle systems (AGVs), one rail-guided vehicle system (RGV), and an automated storage/retrieval system (AS/RS), where each system has one mobile transportation machine installed. Each machine is confined

to the designated working area, as shaded in the figure. Additionally, a few pick-up/drop-off points (P/D points for short) are assigned in each area; these are the only points where the AHMS can pick up or drop off materials. Finally, transfer facilities, such as conveyors, are installed between AHMS systems via paired output and input ports, and they are represented as links in Fig. 3-1. Note that each transfer facility can also serve as a “temporary” buffer with a limited capacity.

In the real scenario, the P/D points for the AGV and RGV represent the I/O ports of manufacturing machines and transfer facilities. For example, the P/D point 1 in the AGV 1 in Fig. 3-1 is an I/O port of a manufacturing machine, and P/D points 2–5 are the I/O ports of transfer facilities. Similarly, the P/D point 9 of the RGV is an I/O port of a manufacturing machine, and the other P/D points are the I/O ports of transfer facilities. Meanwhile, for the AS/RS, the P/D points can also represent buffer storage (refer to the I/O ports 5–16 in the AS/RS in Fig. 3-1).

When a material is to be stored in the AS/RS, a transportation request is generated to move the material to the P/D point of the AS/RS, which represents buffer storage.

As shown in Fig. 3-1, there are two transportation requests to be completed at the moment. Request 1 wants to move the material from the lower right corner (denoted by the blue star) to the upper left corner (the red star). This can be handled by the blue path: AGV1 [port 1 → 4] → RGV [port 7 → 2] → AGV3 [port 3 → 1], where the material *sequentially* crosses AGV1, RGV, and AGV3 AHMSs. Similarly, Request 2 can be handled by the dark path via RGV [port 9 → 6] → AGV 2 [3 → 1]. We call each pair of two P/D points in an AMHS [port i → j] a delivery operation, which is the actual material movement conducted by the



AMHS. In this case, both requests will need the RGV AHMS for transportation.

But if the RGV can only carry one material load at a time, there is a decision to make to carry which material first. Moreover, this decision will affect other delivery operations as well. Therefore, there are five delivery operations to be scheduled to complete the requests in this example.

Figure 3-2 shows another possible path for Request 1: AGV1 [port 1 → 2] → AS/RS [port 1 → 4] → RGV [port 3 → 2] → AGV 3 [port 3 → 1], which consists of four delivery operations. Since both paths meet the job request but differ in the operation sequences, this also offers an opportunity for optimization to minimize the completion time.

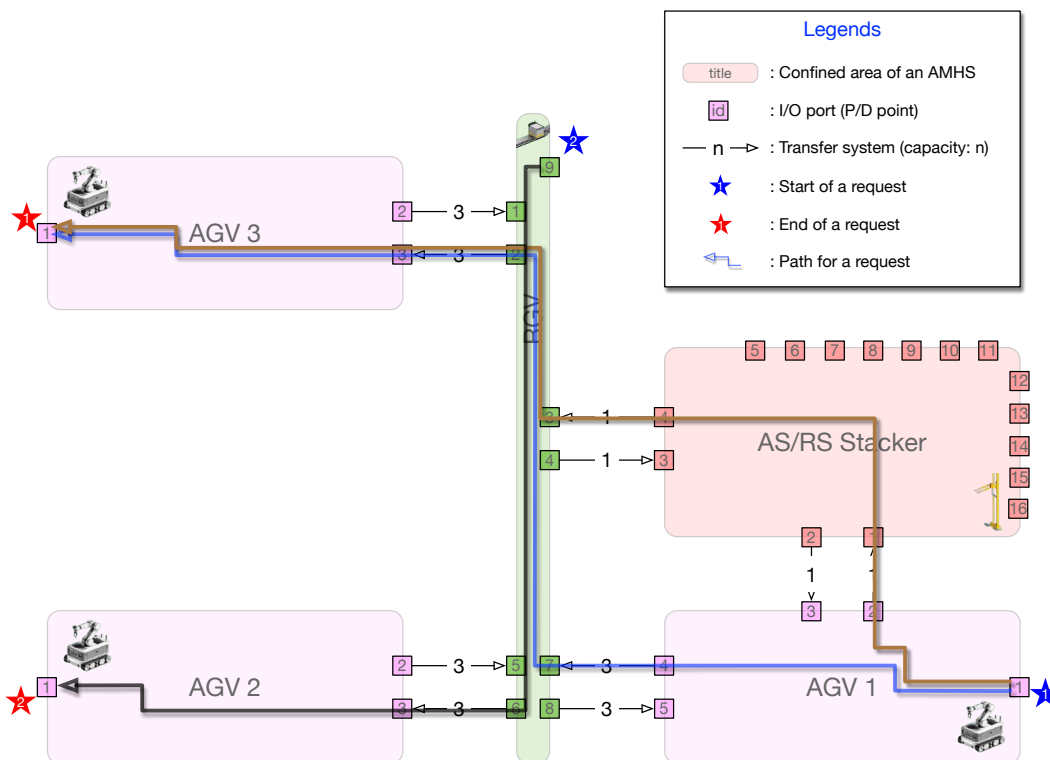
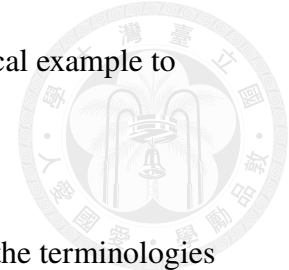


Figure 3-2. An illustration of multiple candidate paths for a request.

This example gives a taste that the core of the MHNSP is subject to candidate path selections and operation sequence scheduling to minimize the

makespan. At the end of this section, we will provide a numerical example to discuss these two decisions further.

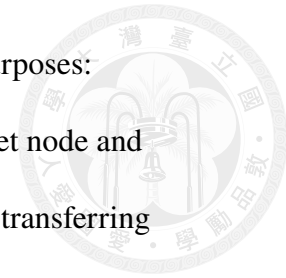


In subsequent sections, we will systematically introduce the terminologies used and elucidate the scope of the problem to complete the definition of the Material Handling Network Scheduling Problem.

3.1.2 Mathematical Formulation

This section depicts the mathematical formulation of the Material Handling Network Scheduling Problem (MHNSP), starting with a few terminology introductions and illustrations of the problem structure. After that, we will define all components of the MHNSP mathematical model, including the optimization goal and constraints.

A *material handling network* (network for short) is a connected structure of multiple material handling systems. Each handling system consists of a unique type of mobile vehicle, robot, or machine that delivers materials within its working area. The transportation equipment can only operate in the designated area and is limited to picking up or dropping off materials at specified locations. These locations are called the pick-up/drop-off points, or P/D points for short. These handling systems and the associated P/D points are modeled as *transportation nodes* of the network. Since each transportation equipment cannot leave its working area, material transfer facilities, such as conveyors, are installed to transfer material between nodes. We denote the transfer facilities as *transfer sites* for generality, which are modeled as directed links from one P/D point of the source



node to a P/D point of the target node. The transfer site serves two purposes: moving the material unidirectionally from the source node to the target node and acting as a storage buffer. Generally, a transfer time is incurred when transferring the material, and since a transfer site only occupies a limited physical space, the buffer capacity is finite. Figure 3-3 models the previous network example as a set of transportation nodes connected with directed transfer sites, which can be regarded as a directed graph with featured vertices and edges.

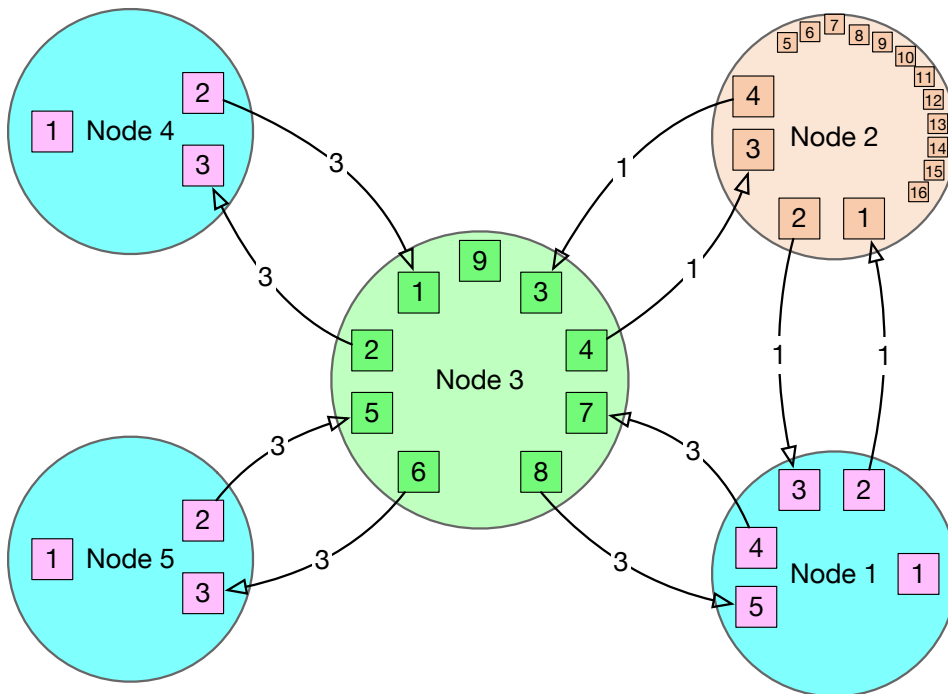


Figure 3-3. Graph representation of the network example.

A material *transportation request* (or job) is usually issued from the manufacturing execution system (MES) to deliver a load of material from a start P/D point to an end one. These points may be located on the same node, and the request can be trivially fulfilled within the node. When the start and end points are located in different nodes, available routing paths should be identified for selection. The set of paths is called *candidate paths*, which can be presented by a sequence of

delivery operations. A *delivery operation* is a material movement from pick-up to drop-off points in the node. Topologically, a transportation request might have no path since the transfer sites connecting nodes are unidirectional. With the *request generated time* given, a transportation job is then defined by the request's start/end points and associated candidate paths, as shown in Fig. 3-4.

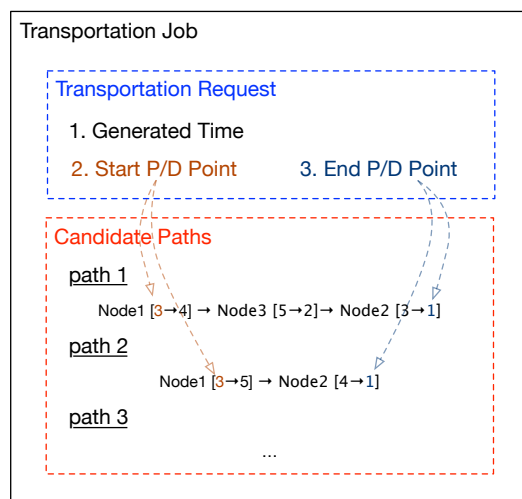
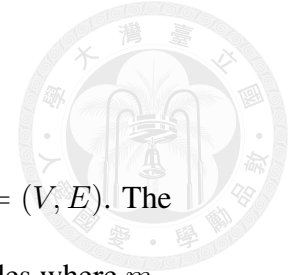


Figure 3-4. Definition of a transportation job.

Optimization Goal

The goal of solving the MHNSP is to generate a schedule that minimizes the maximal completion time (makespan) of the given transportation jobs. This involves selecting an optimal path for each transportation job and coordinating delivery operations for the best job execution schedule.

Following this introduction, we will present the mathematical formulation for the MHNSP. The mathematical model depicts the material handling network, the transportation jobs, and the optimization objective.



Material Handling Network

A material handling network is depicted as a directed graph $G = (V, E)$. The vertices set $V = \{1, 2, \dots, m\}$ consists of indexed transportation nodes where m is the number of nodes. The edge set $E = \{e_1, e_2, \dots, e_r\}$ is the transfer site set, and r is the number of sites installed in the network linking two adjacent nodes.

Let node i have κ_i P/D points located within its encompassing area, assuming that $\kappa_i \geq 1$ for no isolated nodes. Note that the network graph must be a connected graph for a practical MHNSP. Let the P/D point set of node i be $P^{(i)} = \{p_1^{(i)}, p_2^{(i)}, \dots, p_{\kappa_i}^{(i)}\}$. Without loss of generality, we call the mobile transportation equipment in the node a *vehicle*. A vehicle might be an AGV, a rail-guided vehicle, a mobile robot, or a stacker crane. Note that node vehicles travel between P/D points of the node, with no trespassing into other nodes. We assume that the moving speed of the vehicle is not affected by the weight of the load and it is deterministic. Let $\tau_{j,j'}^{(i)}$ be the vehicle moving time from P/D points $p_j^{(i)}$ to $p_{j'}^{(i)}$. For the transportation node i , the transportation time matrix is

$$M^{(i)} = \left[\tau_{j,j'}^{(i)} \right]_{\kappa_i \times \kappa_i}, \text{ where } \tau_{j,j'}^{(i)} \geq 0.$$

A transfer site transfers a material load from the source node P/D point to a point of the target node. Therefore, the transfer site is defined as an ordered P/D point pair $e_k = (p_{b_k}^{(a_k)}, p_{b'_k}^{(a'_k)})$. Note that $p_{b_k}^{(a_k)} \in P^{(a_k)}$ and $p_{b'_k}^{(a'_k)} \in P^{(a'_k)}$, where $a_k, a'_k \in V$, $a_k \neq a'_k$, $b_k = \{1, 2, \dots, \kappa_{a_k}\}$ and $b'_k = \{1, 2, \dots, \kappa_{a'_k}\}$. When a node vehicle delivers a material load to a transfer site, a buffer space is occupied to let the vehicle drop off the load and leave immediately. Let ω_k be the buffer size of site e_k . When all buffers are occupied, the vehicle cannot drop off the material load and

turns “blocked” at the source point of the transfer site. In real applications, the transfer site might work in a First-In-First-out manner or is operated by additional resources. However, we only focus on modeling the behaviors of material transferring and buffer capacity. In our definition, we simply assume that a material load takes at least $\beta_k \geq 0$ transfer time in the transfer site e_k to transfer from the source point to the target point and will cost a unit of the ω_k buffers when the load is begin processed in the site.

Transportation Job

For a given material handling network, the MHNSP should have different sets of transportation jobs to be scheduled (solved) for the problem. Let the transportation job set be $J = \{J_1, J_2, \dots, J_n\}$, where n is the total number of jobs. As mentioned, job J_k is defined as a 4-tuple:

$J_k \equiv \left(g_k, p_{\zeta_k}^{(\xi_k)}, p_{\zeta'_k}^{(\xi'_k)}, \Pi_k \right)$, $k = 1, 2, \dots, n$. In this tuple, g_k is the request generated time; $p_{\zeta_k}^{(\xi_k)}$ is the start point of the job located in node ξ_k , where $\zeta_k \in \{1, 2, \dots, \kappa_{\xi_k}\}$; $p_{\zeta'_k}^{(\xi'_k)}$ is the end point in node ξ'_k , where $\zeta'_k \in \{1, 2, \dots, \kappa_{\xi'_k}\}$.

Let $\Pi_k = \{ \pi_1^{(k)}, \pi_2^{(k)}, \dots, \pi_{\sigma_k}^{(k)} \}$ be the set of candidate paths for the job J_k , where σ_k is the number of candidate paths. As illustrated in Section 3.1.1, an execution path for a job can be represented by a sequence of delivery operations that are executed one after the other. For a general path π , let the operation sequence be $\pi \equiv \langle \rho_1, \rho_2, \dots, \rho_\lambda \rangle$, where λ is the number of operations. We define an operation ρ as a 3-tuple consisting of its executing node η , the pick-up point index θ , and the drop-off point index δ in the node. That is

$$\rho \equiv (\eta, \theta, \delta), \eta \in V; \theta, \delta \in \{1, 2, \dots, \kappa_\eta\}$$



Notice that in a path, every delivery operation except the first operation picks up the load from a target P/D point of a transfer site.

In solving the MHNSP, we need to coordinate the execution sequence of the operations selected and assigned to each node. However, there are some circumstances where the next operation cannot be executed immediately after the end of the previous one. Suppose operations $\rho_j = (\eta, \theta_j, \delta_j)$ and $\rho_{j+1} = (\eta, \theta_{j+1}, \delta_{j+1})$ are two successive operations to be executed by the vehicle of node η . When $\theta_{j+1} \neq \delta_j$, the vehicle needs to travel to the pick-up point $p_{\theta_{j+1}}^{(\eta)}$ after the last dropping off at point $p_{\delta_j}^{(\eta)}$. Consequently, additional travel time is incurred between these operations, which is called the *empty-car moving time* as the vehicle carries no load. This no-value-added movement is often referred to as the *sequence-dependent setup time* in the literature, and the waste is due to the physical constraints of the resource. Figure 3-5 is a Gantt chart example that shows the empty-car moving time between two consecutive delivery operations and at the beginning of the first delivery operation, where the vehicle needs to move from the initial point to the pick-up point of the first operation. For example, the vehicle in node 8 starts from P/D point 1 and moves to P/D points 26, 25, 4, 9, 19, 9, 22, and 9, sequentially.

Let $\pi \equiv \langle \rho_1, \rho_2, \dots, \rho_\lambda \rangle = \langle (\eta_1, \theta_1, \delta_1), (\eta_2, \theta_2, \delta_2), \dots, (\eta_\lambda, \theta_\lambda, \delta_\lambda) \rangle$ be a valid path in set Π_k for job $J_k \equiv \left(g_k, p_{\zeta_k}^{(\xi_k)}, p_{\zeta'_k}^{(\xi'_k)}, \Pi_k \right)$. Since the first operation should pick up the load from the start point of the job, we have the job start point $p_{\zeta_k}^{(\xi_k)} = p_{\theta_1}^{(\eta_1)}$, the pick-up point of the first operation. Similarly, the last operation will deliver the load to the end point of the job, i.e., $p_{\zeta'_k}^{(\xi'_k)} = p_{\delta_\lambda}^{(\eta_\lambda)}$. Notice that each

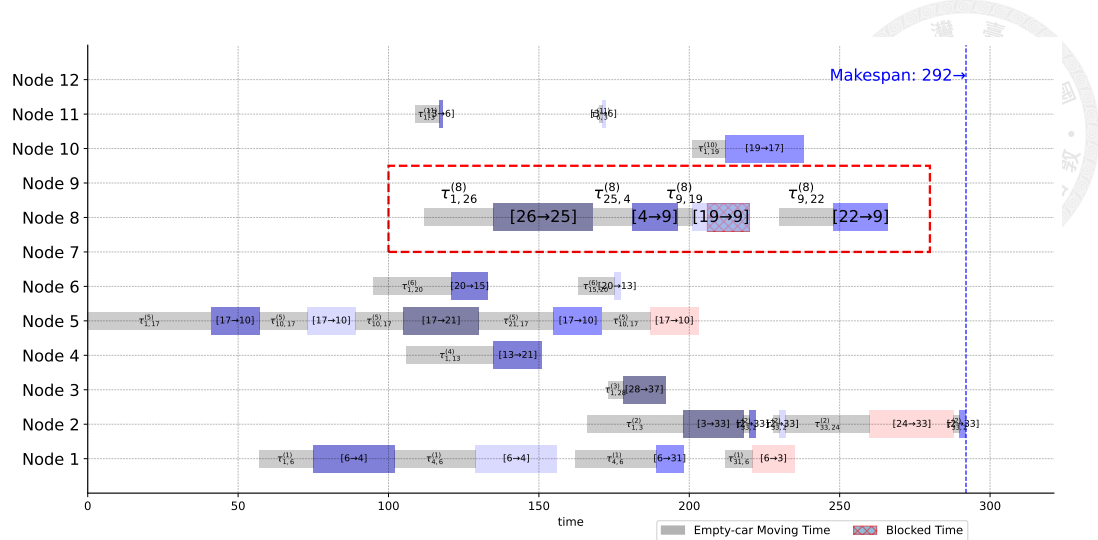


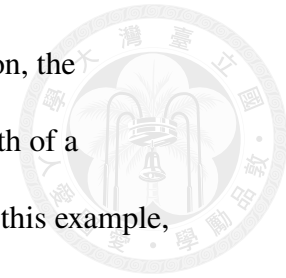
Figure 3-5. Illustration of empty-car moving time between delivery operations.

node is visited at most once in path π ; no cycles are allowed, i.e.,

$$\eta_j \neq \eta_{j'}, \forall j, j' \in \{1, 2, \dots, \lambda\}, j \neq j'. \text{ Therefore, operation number } \lambda \leq m.$$

Moreover, the material must be handed over to the next node via a transfer site. Therefore, the pair of drop-off and pick-up points of two consecutive operations $(p_{\delta_j}^{(\eta_j)}, p_{\theta_{j+1}}^{(\eta_{j+1})})$ must be a transfer site, i.e., $(p_{\delta_j}^{(\eta_j)}, p_{\theta_{j+1}}^{(\eta_{j+1})}) \in E, j = 1, 2, \dots, \lambda - 1$. Let $\epsilon_j \equiv (p_{\delta_j}^{(\eta_j)}, p_{\theta_{j+1}}^{(\eta_{j+1})})$ represent the transfer site connecting operations ρ_j and ρ_{j+1} . When the delivery operation ρ_j is completed, the transfer site ϵ_j will conduct a *transfer operation* to move material to the target point at the target node. If $\epsilon_j = e_l$, the transfer operation takes at least β_l time and is constrained by the buffer size ω_l .

So far, we know that the actual movement of a transportation job is completed by the alternating sequence of delivery operations and transfer sites (operations) $\rho_1 \mapsto \epsilon_1 \mapsto \rho_2 \mapsto \epsilon_2 \mapsto \dots \mapsto \epsilon_{\lambda-1} \mapsto \rho_\lambda$ to reach the end P/D point. In this sequence, ρ_j is the delivery operation, and ϵ_j is the transfer site executing the succeeding transfer operation of delivery operation ρ_j . Figure 3-6 shows a job



scheduling example with 5 transportation jobs. For a transfer operation, the minimum transfer time block is shaded darker. As indicated, the length of a transfer operation must be longer than the minimum transfer time. In this example, there are 3 delivery operations and 2 transfer sites (operations) alternating in path $\pi_2^{(5)}$ for job J_5 .

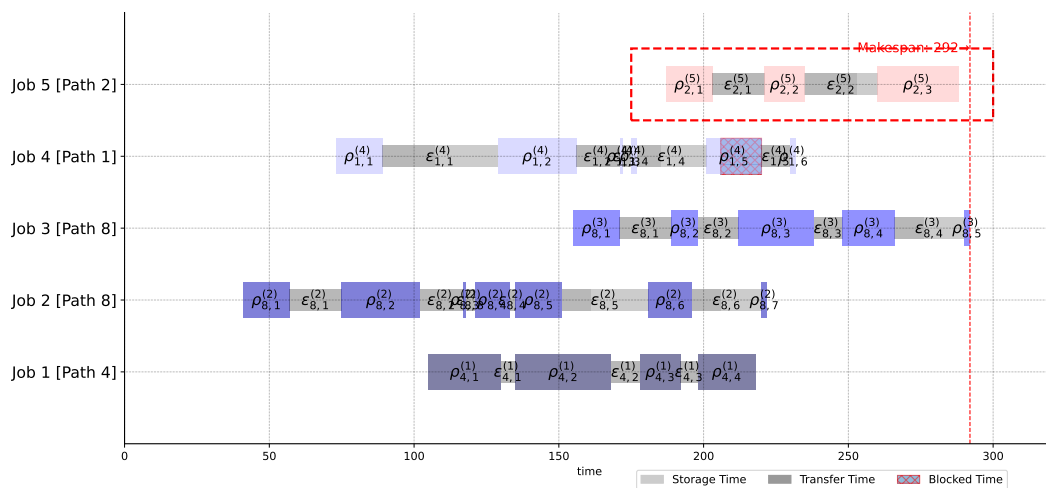
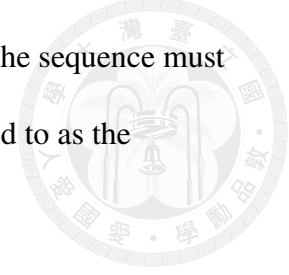


Figure 3-6. Illustration of operation sequence in jobs.

In scheduling job J_k , let the start time be $^s t_k$, which is when the material is picked up at its start point $p_{\zeta_k}^{(\xi_k)}$, and the completion time be $^c t_k$ when the load is delivered to its endpoint $p_{\zeta'_k}^{(\xi'_k)}$. Note that the start time must be later than the request generated time, i.e., $^s t_k \geq g_k$.

In general, a material handling job J_k in the network G can be accomplished via more than one path, i.e., $\sigma_k \geq 1$. If one candidate path is chosen or predetermined for each job ($\sigma_k = 1$) for some reason, the job is named a fixed transportation job; otherwise, it is flexible. If all jobs are fixed, we will have a fixed MHNSP; otherwise, we will have a flexible MHNSP by default. In either case, each job J_k can only be executed by the operation sequence of the selected

candidate path π_k^* from the set Π_k . Moreover, all operations in the sequence must be executed sequentially following their orders, which is referred to as the *operation precedence constraints*.



3.1.3 Data Structure of a Solution

To solve an MHNSP, we must select a candidate path for each job and coordinate sequences of the yielded delivery operations for node vehicles. Specifically, a solution to the problem is setting the start times for all selected operations subject to job operation precedence constraints and vehicle routing sequencing constraints.

Assume that a solution to the problem has selected path π_k^* from the candidate set Π_k for job J_k and $\pi_k^* \equiv \langle \rho_{k,1}^*, \rho_{k,2}^*, \dots, \rho_{k,\lambda_k^*}^* \rangle$, where λ_k^* is the number of delivery operations of the selected path for the job. Also, $\langle \epsilon_{k,1}^*, \epsilon_{k,2}^*, \dots, \epsilon_{k,\lambda_k^*-1}^* \rangle$ is the corresponding transfer sites visited.

The solution needs to set a start time for each delivery operation and transfer operation. Let $t_{k,i}^*$ be the scheduled start time of delivery operation $\rho_{k,i}^*$; similarly, let $\tilde{t}_{k,i}^*$ be the scheduled start time of transfer operation in site $\epsilon_{k,i}^*$. Finally, the start time of the job J_k is set by $^s t_k = t_{k,1}^*$, and the completion time is equal to the start time of the last operation plus its delivery time, that is $^c t_k = t_{k,\lambda_k^*}^* + \tau_{j'',j'''}^{(\eta')}$, where $\eta' = \eta_{k,\lambda_k^*}^*$, $j'' = \theta_{k,\lambda_k^*}^*$, and $j''' = \delta_{k,\lambda_k^*}^*$.



Optimization Goal

The goal of the MHNSP is to find a solution that minimizes the makespan

C^{\max} , i.e.,

$$\min C^{\max} = \min \max_{k=1,2,\dots,n} c t_k$$

This objective aims to minimize the longest completion time among all transportation jobs to enhance the material handling efficiency of the manufacturing execution system.

Summary

In this subsection, we have presented a mathematical model for the MHNSP.

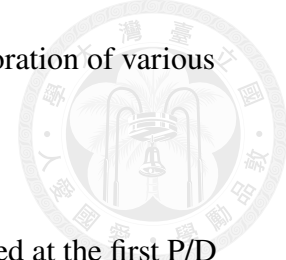
In the next section, we will discuss the assumptions imposed on the problem and the research scope of the problem.

3.1.4 Assumptions and Problem Scope

Since the general model of MHNSP described in the previous section is complicated, some assumptions are needed to simplify the problem so that it can be solved using our methods.

Firstly, we limit our discussion to networks with nodes, each containing only one vehicle. Additionally, the vehicle's load capacity is unit-loaded, meaning that a vehicle can transport only one material load at a time. In some actual applications, multiple vehicles run in the same node, and traffic drawbacks should be considered. Furthermore, heterogeneous vehicles with different moving speeds and

loading capacities might be employed as well. For further exploration of various node types for different AMHSs, please refer to Appendix A.

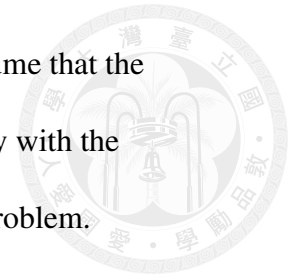


Secondly, it is assumed that each vehicle is initially located at the first P/D point of its executing node. This assumption ensures that the same initial condition is applied to different methods for fair result comparisons.

Thirdly, when a vehicle delivers a load to the drop-off point (the source point of a transfer site) while the site buffers are full, the vehicle and the load turn into a *blocking* state. The blocking state ends when a vacancy is yielded from removing a buffered load by the target node vehicle. Note that a blocked vehicle cannot finish the current operation since the material load cannot be removed from the vehicle to enter the transfer site. If the problem has no capacity limits on all transfer sites, namely $\omega_d = \infty$, no blocking state will occur. In this case, a vehicle ends a delivery operation right at the arrival of the drop-off point and heads for the next delivery operation immediately.

Lastly, we assume the network configuration is well constructed as a connected graph. Since the transfer sites are unidirectional links, not every pair of start and end P/D points has a path between them. This is normal for a network with specified “source” and “sink” nodes, where no jobs transport from a sink node to a source node. We, therefore, assume that the jobs given in the problem are all feasible jobs with at least one candidate path for material delivery. It is also assumed that the P/D points in a node are fully connected, and the transportation times between any two points are calculated and stored in a from-to matrix. In addition, each P/D point can only be assigned to, at most, one transfer site due to

physical limitations. Furthermore, without loss of generality, we assume that the transfer times and the from-to times are all integers. This is to comply with the restriction of using constraint programming techniques to solve the problem.



Fixed and Flexible MHNSP

We define two types of MHNSPs by the number of candidate paths for each job. If every job is given with exactly one candidate path (i.e., $\sigma_k = 1, k = 1, 2, \dots, n$), we call it a fixed MHNSP. Conversely, suppose the problem only specifies each job's start and end points, whose candidate paths can be identified from the network. In this case, the problem is a flexible MHNSP. However, we do not consider all possible candidate paths for a job, instead, only a subset of them are used in our solving methods. We will discuss the heuristics to select the subset of candidate paths in Section 3.2.3.

3.1.5 Summary

Before wrapping up Section 3.1, we revisit a similar sample network introduced at the beginning of this chapter to show a numerical example of the problem

Following the notations defined in Section 3.1.2, we can construct a mathematical model for the network shown in Fig. 3-7. The node set $V = \{1, \dots, 5\}$, and the numbers of P/D points are $\kappa_1 = 5, \kappa_2 = 4, \kappa_3 = 9$,

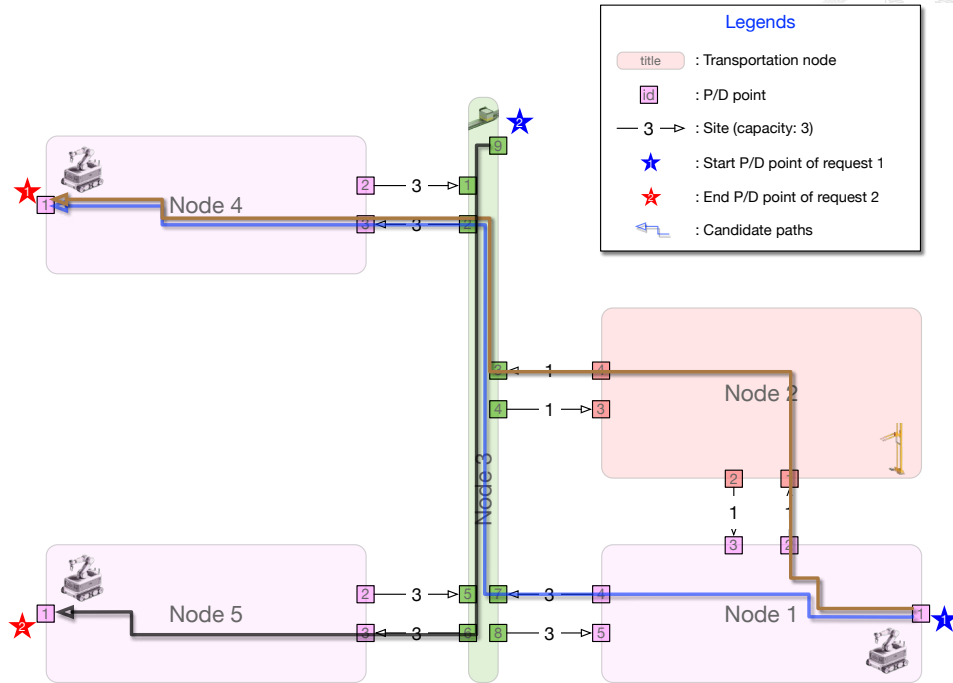


Figure 3-7. A small example of the MHNSP.

$\kappa_4 = 3$, and $\kappa_5 = 3$. Assume the from-to-time matrices for the nodes are

$$M^{(1)} = \begin{bmatrix} 0 & 10 & 15 & 30 & 35 \\ 10 & 0 & 5 & 20 & 25 \\ 15 & 5 & 0 & 15 & 20 \\ 30 & 20 & 15 & 0 & 5 \\ 35 & 25 & 20 & 5 & 0 \end{bmatrix}, \quad M^{(2)} = \begin{bmatrix} 0 & 5 & 20 & 25 \\ 5 & 0 & 15 & 20 \\ 20 & 15 & 0 & 5 \\ 25 & 20 & 5 & 0 \end{bmatrix},$$

$$M^{(3)} = \begin{bmatrix} 0 & 5 & 15 & 20 & 30 & 35 & 30 & 35 & 5 \\ 5 & 0 & 15 & 15 & 25 & 30 & 25 & 30 & 10 \\ 15 & 15 & 0 & 5 & 15 & 10 & 15 & 20 & 20 \\ 20 & 15 & 5 & 0 & 10 & 15 & 10 & 15 & 25 \\ 30 & 25 & 15 & 10 & 0 & 5 & 0 & 5 & 35 \\ 35 & 30 & 10 & 15 & 5 & 0 & 5 & 0 & 40 \\ 30 & 25 & 15 & 10 & 0 & 5 & 0 & 5 & 35 \\ 35 & 30 & 20 & 15 & 5 & 0 & 5 & 0 & 40 \\ 5 & 10 & 20 & 25 & 35 & 40 & 35 & 40 & 0 \end{bmatrix},$$

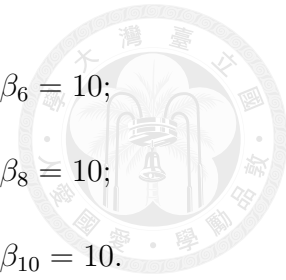
$$M^{(4)} = M^{(5)} = \begin{bmatrix} 0 & 20 & 20 \\ 20 & 0 & 5 \\ 20 & 5 & 0 \end{bmatrix}$$

The set of transfer sites $E = \{e_1, e_2, \dots, e_{10}\}$. Let the definitions of transfer

sites, buffer sizes, and transfer times be

$$e_1 \equiv (p_2^{(1)}, p_1^{(2)}), \omega_1 = 1, \beta_1 = 10; \quad e_2 \equiv (p_2^{(2)}, p_3^{(1)}), \omega_2 = 1, \beta_2 = 10;$$

$$e_3 \equiv (p_4^{(3)}, p_3^{(2)}), \omega_3 = 1, \beta_3 = 10; \quad e_4 \equiv (p_4^{(2)}, p_3^{(3)}), \omega_4 = 1, \beta_4 = 10;$$



$$e_5 \equiv (p_4^{(1)}, p_7^{(3)}), \omega_5 = 3, \beta_5 = 10; \quad e_6 \equiv (p_8^{(3)}, p_5^{(1)}), \omega_6 = 3, \beta_6 = 10;$$

$$e_7 \equiv (p_2^{(3)}, p_3^{(4)}), \omega_7 = 3, \beta_7 = 10; \quad e_8 \equiv (p_2^{(4)}, p_1^{(3)}), \omega_8 = 3, \beta_8 = 10;$$

$$e_9 \equiv (p_6^{(3)}, p_3^{(5)}), \omega_9 = 3, \beta_9 = 10; \quad e_{10} \equiv (p_2^{(5)}, p_5^{(3)}), \omega_{10} = 3, \beta_{10} = 10.$$

The sample problem has two transportation jobs, which are requested at time

0. The job set is $J = \{J_1, J_2\}$; Job $J_1 \equiv (0, p_1^{(1)}, p_1^{(4)}, \Pi_1)$ and $\Pi_1 = \{\pi_1^{(1)}, \pi_2^{(1)}\}$,

where $\begin{cases} \pi_1^{(1)} \equiv \langle (1, 1, 4), (3, 7, 2), (4, 3, 1) \rangle \\ \pi_2^{(1)} \equiv \langle (1, 1, 2), (2, 1, 4), (3, 3, 2), (4, 3, 1) \rangle \end{cases}$. Similarly, job

$J_2 \equiv (0, p_9^{(3)}, p_1^{(5)}, \Pi_2)$ and $\Pi_2 = \{\pi_1^{(2)}\}$, $\pi_1^{(2)} \equiv \langle (3, 9, 6), (5, 3, 1) \rangle$.

Several solutions for this problem may be generated by selecting a candidate path for each job and permuting operation execution orders on nodes. Assume that Solution A selects path $\pi_1^{(1)} \equiv \langle (1, 1, 4), (3, 7, 2), (4, 3, 1) \rangle$ for job J_1 , and $\pi_1^{(2)} \equiv \langle (3, 9, 6), (5, 3, 1) \rangle$ for job J_2 . In total, five operations are to be executed, with one for each node 1, 4, and 5, while node 3 takes two operations. In this solution, let node 3 execute operation (3, 7, 2) first, then (3, 9, 6). The schedule resulting from Solution A is depicted in Fig. 3-8, where the makespan is 145.

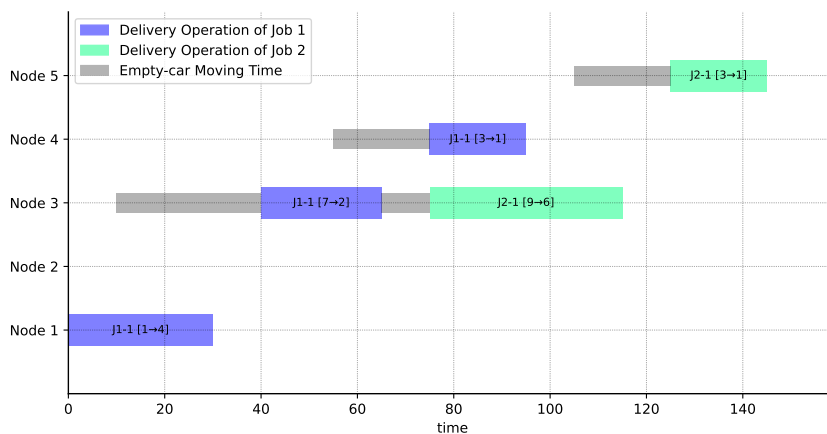


Figure 3-8. Gantt chart of Solution A for the MHNSP sample.

Conversely, assume Solution B lets node 3 execute operation (3,9,6) first,

then (3,7,2). The resulting schedule is shown in Fig. 3-9, and the makespan is 105.

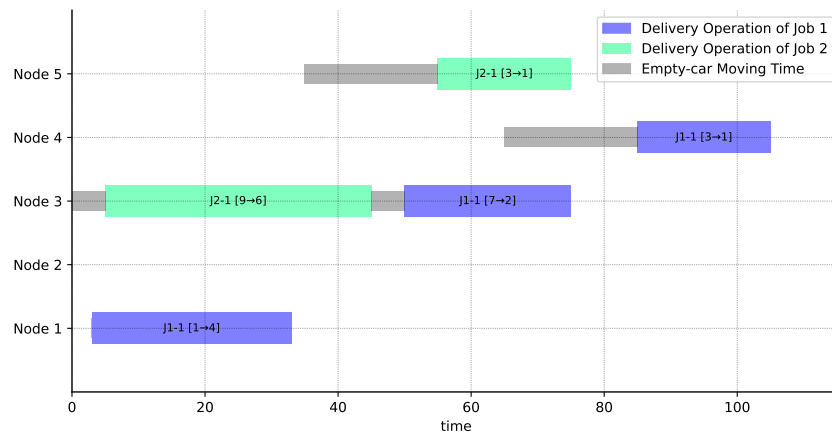


Figure 3-9. Gantt chart of Solution B for the MHNSP sample.

In contrast, assume that Solution C selects the different path

$\pi_2^{(1)} \equiv \langle (1, 1, 2), (2, 1, 4), (3, 3, 2), (4, 3, 1) \rangle$ for job J_1 . This yields six operations that need to be executed. Nodes 1, 2, 4, and 5 each take one operation, and nodes 3 take operations (3,3,2) and (3,9,6). Similarly, let node 3 execute operation (3,3,2) first, which yields the schedule shown in Fig. 3-10, with a makespan of 150.

Instead, let node 3 execute operation (3,9,6) first to generate Solution D, whose schedule is shown in Fig. 3-11. The makespan of solution D is 100.

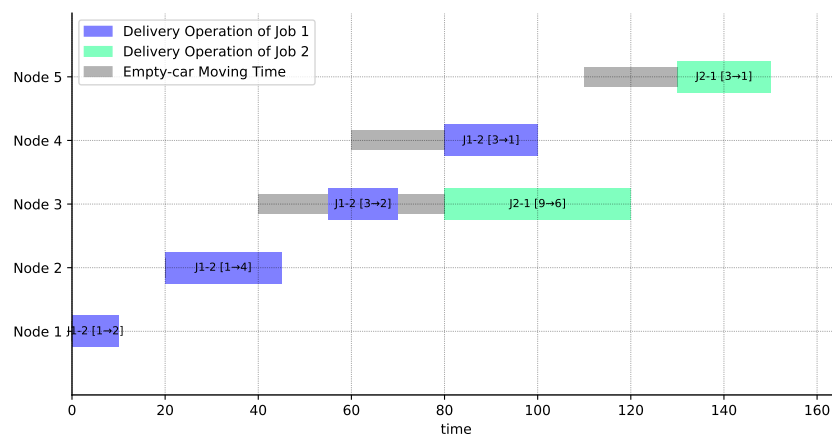


Figure 3-10. Gantt chart of Solution C for the MHNSP sample.

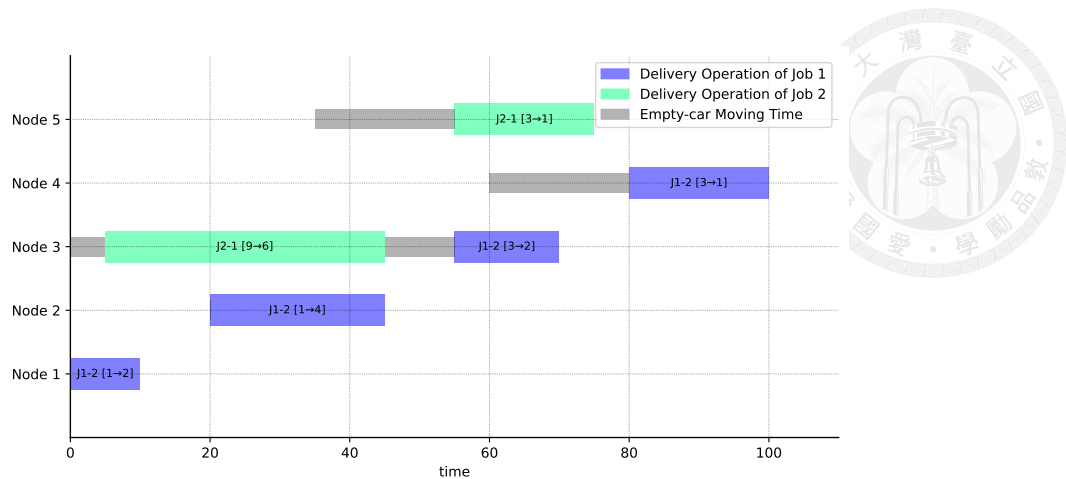


Figure 3-11. Gantt chart of Solution D for the MHNSP sample.

The above example shows that the optimal solution is obtained when job J_1 selects the path $\pi_2^{(1)}$ and node 3 executes (3,9,6) first. Interestingly, when job J_2 is not presented, the makespan is 95 if $\pi_1^{(1)}$ is selected and 100 if $\pi_2^{(1)}$ is selected. This highlights the importance of path selection in the MHNSP when multiple jobs are presented. If one only considers the fastest/shortest path for each job, this would usually yield a suboptimal solution. Furthermore, the execution order also has a significant impact on makespan. Comparing Solutions C and D, we can see that the only difference is the execution order in node 3; yet one yields the worst solution, while the other yields the best. Moreover, it can be found that path selection and operation sequencing have an interaction effect on the yielded makespan. In conclusion, this example depicts the global perspective of path selection and operation sequencing. The optimal solution can only be obtained when both decisions are considered simultaneously.

In this section, we provide a mathematical model to describe the MHNSP and conclude with an example to show the importance of decision-making in solving the MHNSP. Next, we will introduce the procedures to generate test

problems for the MHNSP that will be used later to evaluate the performance of proposed solving methods for the MHNSP.

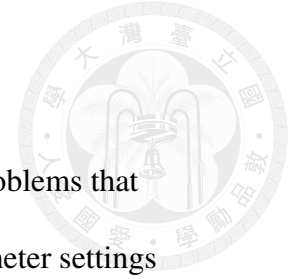


3.2 Numerical Test Problem Generation

This section describes the procedure for creating test problems for the Material Handling Network Scheduling Problem to evaluate the performance of our models and for future research. Each problem describes a material handling network and a set of transportation jobs to be scheduled. Specifically, each problem is stochastically constructed using user-defined parameters relating to the structure complexity and scale-related parameters for job set creations.

After constructing the network and generating the jobs, we pre-select a subset of all possible candidate paths for each job. In the flexible MHNSP, we consider all paths within the pre-selected subset. In contrast, for the fixed MHNSP, we choose a single path from the subset for each job. It is important to note that, for both types of problems, the listed candidate paths must be valid paths capable of completing the job transportation within the network.

The following subsections will detail the procedures for creating a problem for the numerical tests. We will present the user-defined parameters for configuring the network and then describe the procedure for constructing the network and generating transportation jobs. After that, we will show our heuristics for candidate path pre-selection. Finally, we will demonstrate a sample numerical test problem file for the MHNSP and our notations for different problem scales.



3.2.1 User-defined Parameters

User-defined parameters are used to generate numerical test problems that have particular network structures and job complexities. These parameter settings help generate any possible network configurations subject to the discussed assumptions.

First, three primary parameters are chosen by the user to specify the complexity and the scale of the problem: the **number of transportation nodes**, m , the **number of transportation jobs**, n , and the **number of transfer sites**, r . Restrictions on values of these parameters are $m \geq 2$ to have a network of AHMSs, $r \geq m - 1$ to be a connected graph, and $n \geq 2$ for a non-trivial problem.

After acquiring the primary parameters, the user sets the upper bound $\bar{\kappa}$ for the number of P/D points in a node. The problem generator will stochastically set the number for each node such that $2 \leq \kappa_i \leq \bar{\kappa}, \forall i \in V$. Similarly, the user specifies the upper bound $\bar{\tau}$ for the generations of the from-to delivery time between P/D points. The generated time should follow

$$0 \leq \tau_{j,j'}^{(i)} \leq \bar{\tau}; \forall j, j' \in \{1, 2, \dots, \kappa_i\}, \forall i \in V.$$

Likewise, user-defined parameters $\bar{\omega}$ and $\bar{\beta}$ are the upper bounds of the generated buffer sizes and transfer times of the transfer sites. Therefore, the generated buffer sizes $0 \leq \omega_l \leq \bar{\omega}$ and transfer times $0 \leq \beta_l \leq \bar{\beta}, l = 1, 2, \dots, r$.

Another upper bound \bar{g} limits the values of job request times, i.e.,

$0 \leq g_k \leq \bar{g}, k = 1, 2, \dots, n$. Note that we assume the scheduling starts at time 0.

The parameter values specified by the user are the upper bounds of discrete uniform distribution, i.e., $\mathcal{U}(\underline{u}, \bar{u})$, where \underline{u}, \bar{u} are lower and upper bounds (inclusive), respectively. With the given lower bounds, the problem generator assigns uniformly distributed integer values for the variables discussed above. Note that uniformly distributed job request times imply that the interarrival times between two successive jobs approximate the exponential distribution $\sim \text{Exp}(\bar{g}/n)$. The user-defined parameters are listed in Table 3-1.

Table 3-1. Summary of user-defined parameters for problem generation.

Parameter	Description
m	Number of transportation nodes
n	Number of transportation requests/jobs
r	Number of transfer sites
$\bar{\kappa}$	Upper bound of number of P/D points in a node
$\bar{\tau}$	Maximum delivery time in a node
$\bar{\beta}$	Upper bound of transfer time
$\bar{\omega}$	Upper bound of site capacity
\bar{g}	Upper bound of request generated time

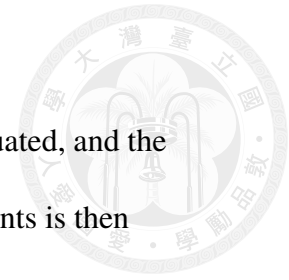
Subsequent sections will present the construction steps of a material handling network, procedures of transportation job generation, and the search algorithm for candidate paths of a generated job.

3.2.2 Network Construction

The first stage in the network construction is to create exactly m nodes and then stochastically set the number of P/D points for each. In order to generate the from-to timetable for each node, dummy 2D coordinates are randomly assigned to

each point j , whose coordinates is (x_j, y_j) . The Manhattan distance $z_{j,j'} = |x_j - x_{j'}| + |y_j - y_{j'}|$ between each pair of points j, j' is evaluated, and the largest distance \bar{z} is identified. The from-to timetable for a pair of points is then scaled from its distance with the time upper bound $\bar{\tau}$ against \bar{z} . Note that since the problem does not consider the actual configuration and connectivity layout of the P/D points in a node, the from-to times can be generated arbitrarily without considering any geometrical restrictions or reachability.

The second stage is constructing transfer sites to connect nodes as a connected graph. We first create a connected subgraph with m transfer site. Then, we bridge the remaining $(r - m + 1)$ sites. Algorithm 1 lists the pseudo-code that constructs the user-defined network in an intended format.



Algorithm 1 NetworkConstruction (m, r)

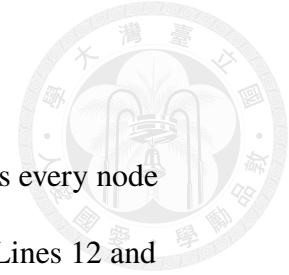
```

1:  $V \leftarrow \{1, 2, \dots, m\}$ 
2: foreach node  $i$  in  $V$  do ▷ Generate P/D points
3:    $\kappa_i \leftarrow \sim \mathcal{U}(2, \bar{\kappa})$ 
4:    $P^{(i)} \leftarrow \{p_j^{(i)} \mid j = 1, 2, \dots, \kappa_i\}$ 
5:    $M^{(i)} \leftarrow [\tau_{j,j'}^{(i)}]_{\kappa_i, \kappa_i}$ 
6:    $(x_j, y_j) \leftarrow (\text{Random}(), \text{Random}()), j = 1, 2, \dots, \kappa_i$ 
7:    $z_{j,j'} \leftarrow |x_j - x_{j'}| + |y_j - y_{j'}|, \forall j, j' \in \{1, 2, \dots, \kappa_i\}$ 
8:    $\bar{z} \leftarrow \max_{\forall j, j'}(z_{j,j'}), \forall j, j' \in \{1, 2, \dots, \kappa_i\}$ 
9:    $\tau_{j,j'}^{(i)} \leftarrow \lceil \bar{\tau} \cdot \frac{z_{j,j'}}{\bar{z}} \rceil, \forall j, j' \in \{1, 2, \dots, \kappa_i\}$ 
10:  $E \leftarrow \{\}; \bar{E} \leftarrow \{\}; \tilde{G} \leftarrow \{\text{RndOne}(V)\}; k \leftarrow 1$ 
11: while  $|\tilde{G}| < m$  do ▷ Create a connected subgraph
12:    $i \leftarrow \text{RndOne}(\tilde{G})$ 
13:    $j \leftarrow \text{RndOne}(V \setminus \tilde{G})$ 
14:   TryCreateTransferSiteFromNodeItoJ( $E, \bar{E}, k, i, j$ )
15:   if  $|\tilde{G}| \neq |E|$  then
16:      $\tilde{G} \leftarrow \tilde{G} \cup \{j\}$ 
17: while  $|E| < r$  do ▷ Create the remaining sites
18:    $i \leftarrow \text{RndOne}(V)$ 
19:    $j \leftarrow \text{RndOne}(V \setminus \{i\})$ 
20:   TryCreateTransferSiteFromNodeItoJ( $E, \bar{E}, k, i, j$ )

```

Note that the operation $\text{Random}()$ in the pseudo-code returns a random real number whose value is within $[0.0, 1.0)$ and the operation $\text{RndOne}(X)$ returns a randomly selected element from the specified set X . Also, \setminus is the set subtraction operator. Expression $X \setminus Y$ removes the common elements of set X and Y , (i.e., $X \cap Y$) from X .

In the pseudo-code listed in Algorithm 1, Lines 6 and 7 randomly allocate a 2D point for each P/D point and then calculate the Manhattan distances between pairs of points. Line 9 scales a distance to map it to an integral traveling time



against the upper limit $\bar{\tau}$.

Lines 11–16 create a minimum spanning subgraph that includes every node $i \in V$. \tilde{G} book-keeps the node in the current constructing subgraph. Lines 12 and 13 randomly select a node not in \tilde{G} and try to construct a transfer site to connect the selected node to the constructing subgraph. When every node is included in the connected subgraph \tilde{G} , there will be exact $m - 1$ transfer sites. Lines 17–20 construct the remaining $(r - m + 1)$ transfer sites. In the algorithm, the function TryCreateTransferSiteFromNodeItoJ is called to establish a transfer site between node i and j if possible. The pseudo-code of the algorithm is shown in Algorithm 2.

Algorithm 2 TryCreateTransferSiteFromNodeItoJ(E, \bar{E}, k, i, j)

- 1: $b_k \leftarrow \sim \mathcal{U}(1, \kappa_i)$
 - 2: $b'_k \leftarrow \sim \mathcal{U}(1, \kappa_j)$
 - 3: **if** $p_{b_k}^{(i)}, p_{b'_k}^{(j)} \notin \bar{E}$ **then**
 - 4: $e_k \leftarrow (p_{b_k}^{(i)}, p_{b'_k}^{(j)}); \omega_k \leftarrow \sim \mathcal{U}(1, \bar{\omega}); \beta_k \leftarrow \sim \mathcal{U}(1, \bar{\beta})$
 - 5: $E \leftarrow E \cup \{e_k\}; \bar{E} \leftarrow \bar{E} \cup \{p_{b_k}^{(i)}, p_{b'_k}^{(j)}\}; k \leftarrow k + 1$
-

Lines 1 and 2 randomly select a P/D point index in each node. Line 3 checks if the selected P/D points have been defined as a transfer site. If not, we create a transfer site and update the transfer site set E and the set of used P/D points \bar{E} .

3.2.3 Transportation Job Generation

In generating the set of transportation jobs, we first assign each job a request time with a discrete uniformly distributed random value

$g_k \leftarrow \sim \mathcal{U}(0, \bar{g}), k = 1, 2, \dots, n$. Subsequently, we randomly select two P/D

points as the start and end points of the job. Then, we execute an operation to find its candidate paths. If no path exists for the pair of points, it is discarded, and re-selection is repeated. The pseudo-code of the procedure is shown in Algorithm 3.

Algorithm 3 TransportationJobGeneration (n)

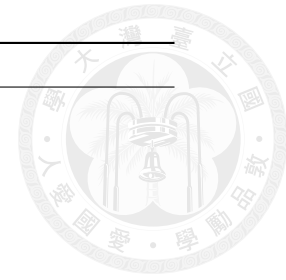
```

1: for  $k \leftarrow 1$  to  $n$  do
2:    $g_k \leftarrow \sim \mathcal{U}(0, \bar{g})$ 
3:    $s \leftarrow \text{RndOne}(V); s' \leftarrow \text{RndOne}(V)$ 
4:    $t \leftarrow \text{RndOne}(\{1, 2, \dots, \kappa_s\}); t' \leftarrow \text{RndOne}(\{1, \dots, \kappa_{s'}\})$ 
5:    $\Pi_k \leftarrow \text{FindCandidatePaths}(s, t, s', t')$ 
6:   if  $\Pi_k = \emptyset$  then
7:     goto 3
8:    $J_k \leftarrow \left( g_k, p_{\zeta_k}^{(\xi_k)}, p_{\zeta'_k}^{(\xi'_k)}, \Pi_k \right) \equiv \left( g_k, p_t^{(s)}, p_{t'}^{(s')}, \Pi_k \right)$ 
9:  $J \leftarrow \{J_k \mid k = 1, 2, \dots, n\}$ 

```

Lines 3 and 4 randomly select a start and an endpoint for job J_k , starting from $p_t^{(s)}$ to $p_{t'}^{(s')}$. However, they are reselected when lines 6 and 7 find no candidate path to deliver the material load. The function $\text{FindCandidatePaths}(s, t, s', t')$ returns the candidate path set for the given start/end points. The pseudo-code of the function is listed in Algorithm 4. The job setting is completed in Line 8 by aggregating the request time, start and end points, and the candidate path set.

Candidate paths for a job are traversed via a recursive procedure, routing from the start point of the job to the end point without cycles. The procedure constructs a path by consecutively appending a feasible delivery operation to it until the end point is reached or no viable operation can succeed.



Algorithm 4 FindCandidatePaths(s, t, s', t')

```

1: if  $s = s'$  then
2:   if  $t = t'$  then
3:     return  $\emptyset$ 
4:   else
5:     return  $\{\langle (s, t, t') \rangle\}$ 
6:  $\Pi \leftarrow \{\}$ ;  $\pi \leftarrow \langle \rangle$ ;  $\rho \leftarrow (s, t, -1)$ 
7: RecursivePathConstruction( $\Pi, \pi, \rho, s', t'$ )
8: return  $\Pi$ 

```

No job is generated if the start and end points are the same since no material handling is required, as indicated in Line 3 of Algorithm 4. Line 5 composes a job with a trivial path of one operation for a job delivering material within a node. Line 7 subsequently constructs all candidate paths by executing a recursive operation, RecursiveConstructPaths(). The operation starts with the half-defined first operation ρ , i.e. $(s, t, -1)$, beginning from the start point $p_t^{(s)}$ in the starting node s , yet the drop-off point is not determined. Then, it traverses all outgoing transfer sites to complete the current operation (the drop-off point) and sets up the successive half-defined operation for the nested recursion. A valid path is then constructed once the recursion reaches the end node. However, an invalid path is discarded when no outgoing transfer sites exist or outgoing sites route back to traversed nodes before reaching the end node. The latter case forms a cycle, not a valid path.

Algorithm 5 lists the pseudo-code of the recursive algorithm that constructs the candidate path set. In Algorithm 5, Lines 1 – 3 identify the path routing reaches the end P/D point to terminate the recursion. The valid path π^* is constructed by cloning the path under traversing, π , and adding the final operation ρ to it, as

shown in Line 2. Note that the operator \oplus appends the operation on the right-hand side to the path on the left-hand side.

Algorithm 5 RecursivePathConstruction($\Pi, \pi, \rho = (\eta, \theta, \delta), s', t'$)

```

1: if  $\eta = s'$  then
2:    $\delta \leftarrow t'$ ;  $\pi^* \leftarrow \pi \oplus (\eta, \theta, \delta)$ ;  $\Pi \leftarrow \Pi \cup \{\pi^*\}$ 
3:   return
4:  $E' \leftarrow \left\{ e_k \equiv \left( p_{b_k}^{(a_k)}, p_{b'_k}^{(a'_k)} \right) \mid a_k = \eta, k \in \{1, 2, \dots, r\} \right\}$ 
5: if  $E' = \emptyset$  then
6:   return
7: foreach site  $e \equiv (p_b^{(\eta)}, p_{b'}^{(\eta')}) \in E'$  do
8:    $\triangleright$  Check if appending the outgoing site will form a loop  $\triangleleft$ 
9:   if  $\eta' \notin \{\tilde{\eta} \mid \tilde{\rho} \equiv (\tilde{\eta}, \tilde{\theta}, \tilde{\delta}), \tilde{\rho} \in \pi\}$  then
10:     $\delta \leftarrow b$ ;  $\pi \leftarrow \pi \oplus (\eta, \theta, \delta)$ ;  $\rho' \leftarrow (\eta', b', -1)$ 
11:    RecursivePathConstruction( $\Pi, \pi, \rho', s', t'$ )
12:     $\pi \leftarrow \pi \ominus \rho$ 

```

Line 4 constructs the set of outgoing transfer sites of the current node η that executes the current operation $\rho \equiv (\eta, \theta, \delta)$. Line 5 terminates the path search recursion since no outgoing transfer site exists to continue the path.

Lines 7–12 loop through all outgoing transfer sites for path-searching trials by confirming the current operation and setting up the succeeding operation for the nested recursion. Therefore, the drop-off point of the current operation is set to the source point of the outgoing transfer site, and the successive operation's pick-up point is the site's target point.

The target node η' of the consecutive operation is subject to a cycling check against the current path π . Line 9 checks if the target node η' appears in the current path π . If not, Line 10 accepts the current operation by setting the drop-off point to

the source point of the transfer site $\delta \leftarrow b$. Then, the path π is appended with the current operation ρ to continue the path via the outgoing transfer site. The corresponding operation is then half-defined, $\rho' \leftarrow (\eta', b', -1)$, for further recursion. After the transfer site trial, Line 12 detaches the appended operation ρ from π where the operator \ominus removes the operation on the right-hand side from the path on the left-hand side. The removal is to resume the original condition of the path in the current recursion for the next transfer site trial.

Next, we will explain how the candidate paths are pre-selected for each job so the problem becomes solvable using our proposed methods.

Candidate Path Pre-selection

Since the number of all possible candidate paths for a job grows exponentially when the number of nodes increases, we adopt heuristics to pre-select the most promising subset of the paths. This procedure is necessary to make a meaningful comparison between our solving methods, which will be introduced in the next chapter.

First, we define the *free-run time* as the minimum required time to complete a candidate path. The value is obtained by summing the process time of each delivery/transfer operation in the path. The procedure is described in Algorithm 6.

**Algorithm 6** GetFreeRunTime(k, l)

```

1:  $t \leftarrow 0$ 
2: for  $i = 1, 2, \dots, \lambda_{k,l}$  do
3:    $\eta \leftarrow \eta_{l,i}^{(k)}; \theta \leftarrow \theta_{l,i}^{(k)}; \delta \leftarrow \delta_{l,i}^{(k)}$ 
4:    $t \leftarrow t + \tau_{\theta,\delta}^{(\eta)}$ 
5:   if  $i < \lambda_{k,l}$  then ▷ If the operation is not the last one
6:      $e_d \leftarrow \epsilon_{l,i}^{(k)}; t \leftarrow t + \beta_d$ 
7: return  $t$  ▷ Return the free-run time

```

Assume that there are total σ_k candidate paths for job J_k . The idea of the pre-selection is to include only the fastest few paths. Let ϕ be the cut-off factor. We include only the paths with free-run time shorter than $\phi \cdot \underline{t}_k$, where \underline{t}_k is the shortest free-run time of all possible candidate paths Π_k for job J_k . However, this pre-selection might result in too few or too many paths. Therefore, we define $\underline{\sigma}$ and $\bar{\sigma}$ as the minimum and maximum number of paths, respectively, to ensure enough path options for each job. The entire pre-selection procedure is shown in Algorithm 7. The operation $\text{Sort}(X, Y)$ in Line 3 sorts the set X by the corresponding values in Y .

Algorithm 7 PathPreselect(Π_k)

```

1:  $\underline{\sigma} \leftarrow 3; \bar{\sigma} \leftarrow 30; \phi \leftarrow 2;$ 
2:  $t_l \leftarrow \text{GetFreeRunTime}(k, l), l = 1, 2, \dots, \sigma_k$ 
3:  $\Pi_k \leftarrow \text{Sort}(\Pi_k, [t_1 t_2 \dots t_{\sigma_k}])$ 
4:  $[t_1 t_2 \dots t_{\sigma_k}] \leftarrow \text{Sort}([t_1 t_2 \dots t_{\sigma_k}])$ 
5:  $\underline{t}_k \leftarrow t_1; \bar{l} \leftarrow 0$ 
6: for  $l = 1, 2, \dots, \sigma_k$  do
7:   if  $(l \leq \underline{\sigma})$  or  $((t_l \leq \phi \cdot \underline{t}_k)$  and  $(l \leq \bar{\sigma}))$  then
8:      $\bar{l} \leftarrow l$ 
9:  $\Pi_k \leftarrow \{\pi_l^{(k)} \mid l \leq \bar{l}, l = 1, 2, \dots, \sigma_k\}; \sigma_k \leftarrow \bar{l}$  ▷ Update the candidate path set

```

For a fixed MHNSP, we assign a path from the pre-selected candidate path



set to each job. In contrast, all the pre-selected candidate paths are considered in a flexible MHNSP.

3.2.4 Problem Scales, Types and File Format

In the previous subsections, we described the procedures to generate problems for numerical tests. In this subsection, we will introduce the problems that will be used in our numerical tests in Chapter 5. Finally, we will provide the sample files of the generated problems.

Problem Scales

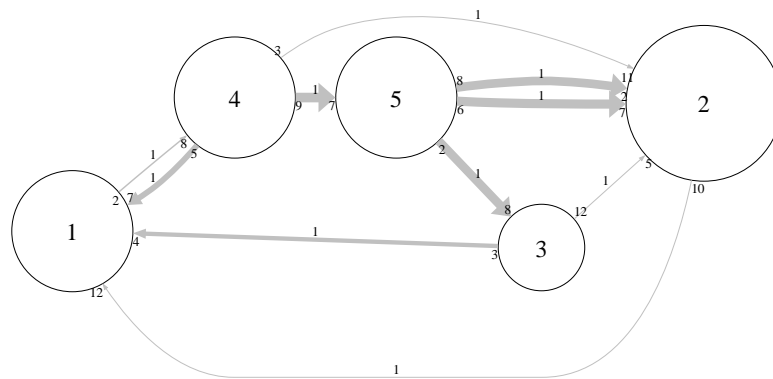
In our research, we categorize the numerical test problems into *four problem scales*: small, medium, large, and extra-large, according to their network complexities and the number of transportation jobs. The user-defined parameters used for generation are summarized in Table 3-2, where the notations of parameters follow those described in Table 3-1. The differences in problem scales are mainly reflected in the number of nodes m , number of jobs n , and number of sites r . Notably, the complexity of the large-scale problems already matches the real-world case of a real display panel factory.

Table 3-2. Parameters for problem generation of different scales.

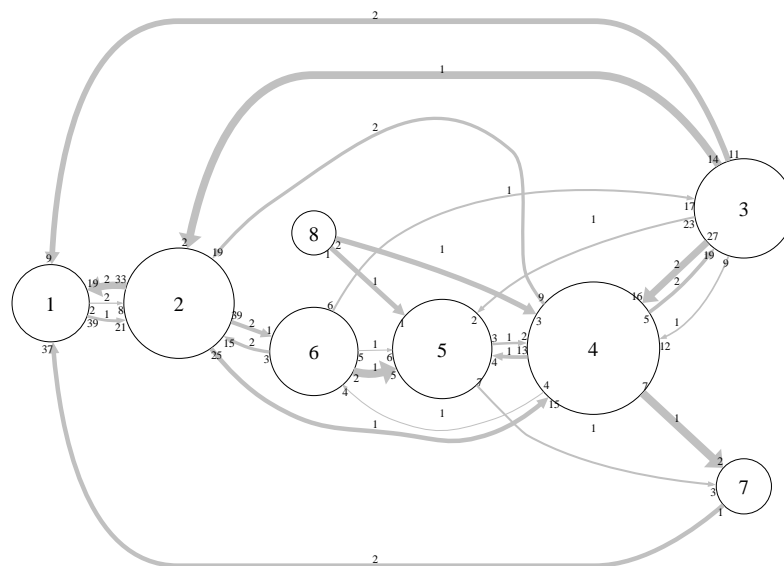
Scale	Notation	m	n	r	$\bar{\kappa}$	$\bar{\tau}$	$\bar{\beta}$	$\bar{\omega}$	\bar{g}
Small	N5S10-J5	5	5	10	20	10	20	1	0
Medium	N8S24-J8	8	8	24	40	10	20	2	0
Large	N12S48-J8	12	8	48	50	10	20	3	0
Extra	N30S100-J20	30	20	100	100	100	20	3	0

We also provide samples of the graph representations for different problem scales in Fig. 3-12. The site capacity and the source/target P/D points of the site

are shown in the middle of the arc and at the head/tail of the arc, respectively. Note that the line width of the arc is inversely proportional to the transfer time in the corresponding site. Finally, the size of the node is proportional to the number of transfer sites connecting to it.

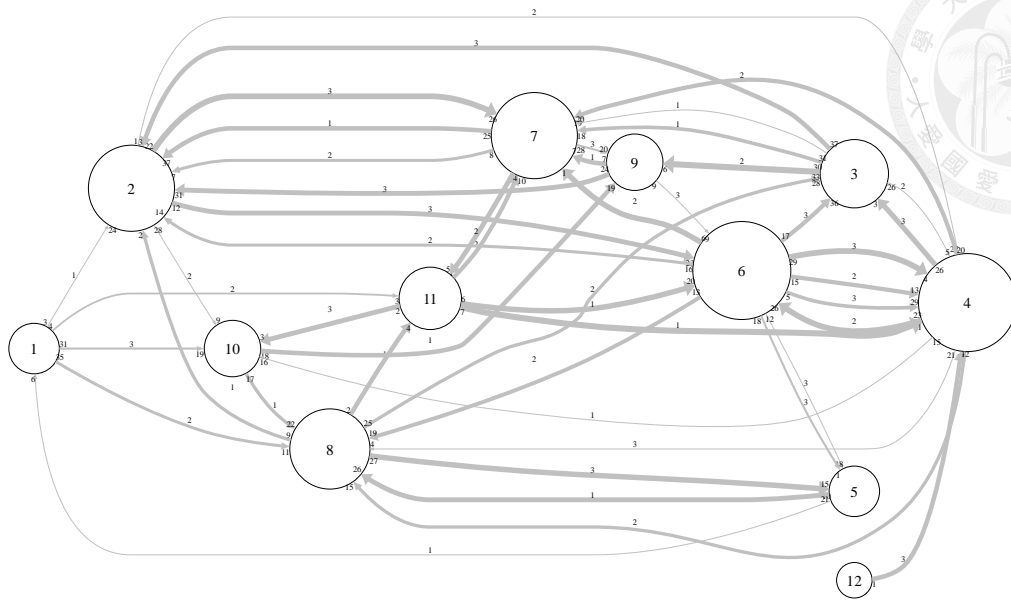


(a) Small-scale network.

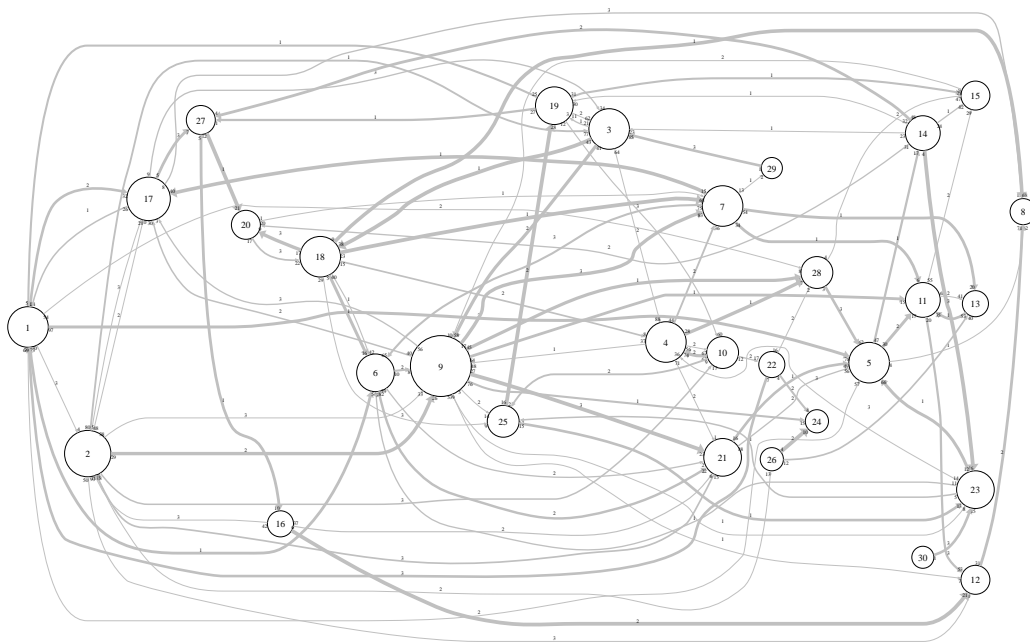


(b) Medium-scale network.

Figure 3-12. Comparison of generated networks in different scales.



(c) Large-scale network.



(d) Extra-large-scale network.

Figure 3-12. Comparison of generated networks in different scales. (cont.)

Problem Types

To conduct more in-depth research, we also consider *five types of problem settings*. These problem types/settings differ in the number of candidate paths for

each job and whether the buffer capacity limit applies.

In a fixed MHNSP, there is only one path chosen from the pre-selected candidate paths Π_k for job J_k . By default, we select the path l_k^* with the smallest free-run for the job J_k , that is

$$l_k^* = \arg \min_{l=1,2,\dots,\sigma_k} \text{GetFreeRunTime}(k, l).$$

Therefore, we have $\Pi_k \leftarrow \{\pi_{l_k^*}^{(k)}\}$ and $\sigma_k \leftarrow 1$. However, we also consider the case of the path being *randomly* selected from the subset, namely

$$\Pi_k \leftarrow \{\text{RndOne}(\Pi_k)\}.$$

While in a flexible MHNSP, all the pre-selected candidate paths are considered.

On the other hand, we suppose that each transfer site has a finite buffer size ω_d that will “block” the vehicle when no vacancy is left. However, some studies in the literature did not consider the site capacity. Therefore, we separate problem cases to determine whether the site capacity constraint is complied with. In the problem type with infinite capacity, the site capacity is ignored; otherwise, the capacity limit is applied.

We use a two-symbol notation to indicate the types of problems. The first symbol indicator if the problem is fixed or flexible. The second symbol indicates if the site capacity is finite. In this research, we study five types of problems: fixed/infinite, flexible/infinite, fixed/finite, fixed-random/finite, and flexible/finite settings. We denote the five types as $1/\infty$, n/∞ , $1/n$, $1'/n$, and n/n , respectively. The comparison of problem types is summarized in Table 3-3.



Table 3-3. Comparison of different problem types.

Problem Type	# of paths	Site Capacity Limit	Path Pre-selection Criteria
1/∞	1	ignored	best
1/n	1	applied	best
n/∞	Multiple	ignored	algorithm
n/n	Multiple	applied	algorithm
1'/n	1	applied	random

File Format

This subsection describes how the numerical test problems are saved and organized in folders. Since various network configurations and job sets can be generated with the same user-defined parameters, we organize and name them according to their problem scales. Figure 3-13 shows the directory structure of all the problems used in our research.

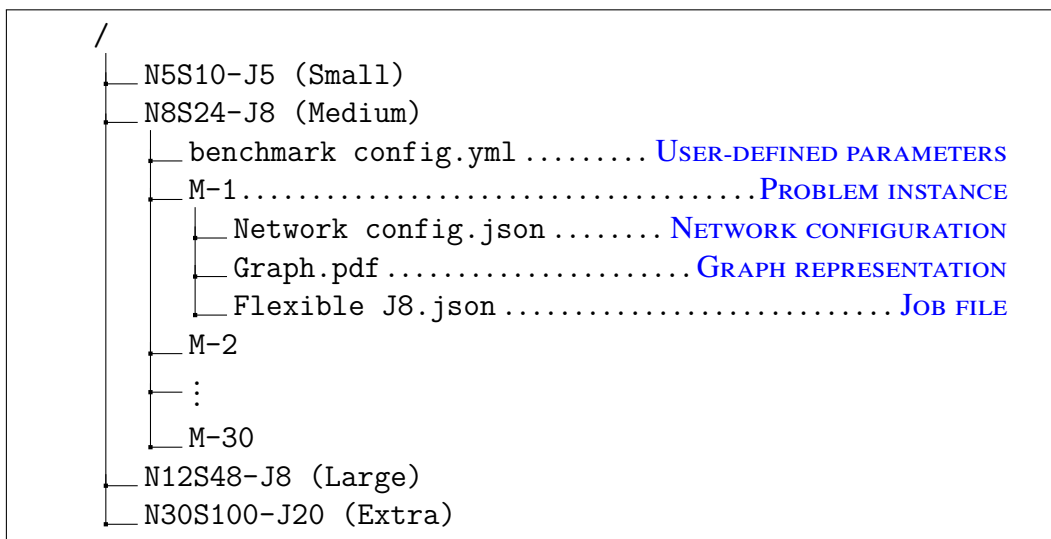
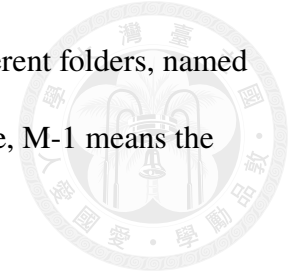


Figure 3-13. Directory structure of the numerical test problem files.

In each problem scale folder, the user-defined parameters used to create the problems of this scale are saved in a file. The context of a sample user-defined parameter file is shown in File 3-1. This file was used to create 30 medium-scale

problem instances. We saved the generated 30 networks in different folders, named with a prefix of the problem scale and a unique ID. For example, M-1 means the first generated network of the medium-scale problem



File 3-1. Problem configuration file for medium-scale problems.

```
# benchmark config.yml
num_networks: 30      # number of networks
num_nodes: 8         # number of nodes
num_pd: 40           # maximum number of P/D points in a node
num_trans_site: 24   # number of sites
max_node_vehicle: 1  # maximum number of vehicles in a node
max_trans_capacity: 2 # maximum capacity of a site
max_trans_pt: 20     # maximum transfer time in a site
max_from_to: 10      # maximum delivery time in a node

num_jobs: 8          # number of jobs
max_start_after: 0   # maximum request generated time
flexible: True       # flexible MHNSP
node_init_pos: 0     # vehicle initial P/D point
```

In each network folder, we saved the network configuration in a JSON file that described the details of the material handling network. A sample network configuration file for network S-1 is shown in File 3-2. Note that the index in the file starts from 0. In the file, we show the from-to timetable for node 5 and omit the others. Finally, each row in the field “transfer sites” lists the details of the site: the source node, the source P/D point ID, the target node, the target P/D point ID, the transfer time, and the site capacity.

Besides the network configuration file, the network graph is displayed in the Graph.pdf file. The schematic diagram is generated by Graphviz. The sample drawings can be found in Fig. 3-12 on page 51.

Finally, we save a set of transportation jobs for each network in a JSON file. File 3-3 is an example of the job file for network S-1. Note that the index in the file starts from 0. Each job has the start node, the start P/D point ID, the end node, and

the end P/D point ID specified. In the file, the “start_after” field is the request generated time of the jobs.



File 3-2. Network configuration file for network S-1.

```
# Network config.json
{
  "num_nodes": 5,
  "num_trans_site": 10,
  "flexible": true,
  "node_init_pos": 0,
  "num_node_veh":
    [1, 1, 1, 1, 1],
  "num_pds_in_node":
    [17, 14, 12, 12, 10],
  "node_from_to":
    [
      ...
      [[0, 22, 22, 21, 5, 14, 21, 16, 18, 8],
       [22, 0, 2, 7, 21, 10, 2, 8, 5, 15],
       [22, 2, 0, 5, 20, 10, 3, 7, 5, 15],
       [21, 7, 5, 0, 20, 12, 6, 5, 7, 15],
       [5, 21, 20, 20, 0, 11, 19, 15, 16, 6],
       [14, 10, 10, 12, 11, 0, 9, 8, 6, 6],
       [21, 2, 3, 6, 19, 9, 0, 7, 3, 14],
       [16, 8, 7, 5, 15, 8, 7, 0, 6, 10],
       [18, 5, 5, 7, 16, 6, 3, 6, 0, 11],
       [8, 15, 15, 15, 6, 6, 14, 10, 11, 0]],
      ...
    ],
  "trans_sites":
    [[0, 1, 3, 7, 18, 1],
     [1, 9, 0, 11, 20, 1],
     [2, 11, 1, 4, 19, 1],
     [2, 2, 0, 3, 12, 1],
     [3, 4, 0, 6, 9, 1],
     [3, 8, 4, 6, 1, 1],
     [3, 2, 1, 10, 19, 1],
     [4, 7, 1, 1, 3, 1],
     [4, 5, 1, 6, 3, 1],
     [4, 1, 2, 7, 3, 1]]
}
```

File 3-3. Job file for network S-1.

```
# Flexible J5.json
{
  "num_jobs": 5,
  "start_after":
    [0, 0, 0, 0, 0],
  "jobs":
    [[3, 0, 1, 0], [0, 9, 2, 3], [2, 6, 0, 5],
     [1, 12, 2, 8], [1, 12, 0, 0]]
}
```

3.2.5 Summary

This section has thoroughly outlined the process of generating numerical test problems for Material Handling Network Scheduling Problems. The approach begins with defining user-defined parameters to shape the network's structure and complexity. We then detailed the steps of constructing a network that adheres to these parameters. After that, we highlighted the processes involved in generating transportation requests and developing an algorithm capable of identifying all optional paths for any given request. In the last subsections, we described the procedure to pre-select the candidate paths and demonstrated some sample files of the generated problems.

Moving forward, the next chapter will delve into various methodologies designed to effectively solve the MHNSP.





Chapter 4

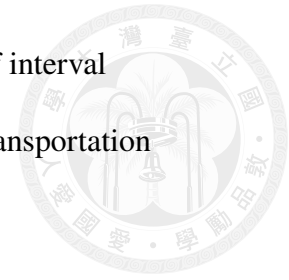
Constraint Programming, Integer Programming, and Differential Evolution Models for Material Handling Network Scheduling Problem

This chapter explores three methodologies for solving the Material Handling Network Scheduling Problem, including *Constraint Programming*, *Integer Programming*, and *Metaheuristic algorithm*. With different modeling principles and solving approaches, each approach offers a distinct procedure to solve the MHNSP. The following sections provide an in-depth analysis of these methodologies, highlighting their procedures for determining the makespan and developing constraints that capture the behaviors of the material handling network system.

4.1 Constraint Programming Model for Solving the Material Handling Network Scheduling Problem

The literature indicates that Constraint Programming (CP) is an effective modeling tool for solving optimization problems with complicated solution structures and logic constraints, particularly resource scheduling problems involving time intervals [4, 12, 16, 25]. In this section, we rigorously derive a CP model for the discussed MHNSP.

The key feature of our CP model is the hierarchical structure of interval variables. This structure makes it easy to derive constraints for our transportation jobs, nodes, and transfer sites.



4.1.1 Job Constraints

This subsection presents the CP model components relating to the transportation job constraints. First, we will have an overview of the proposed hierarchical structure and explain the merits of adopting this structure. Then, we will formally define the interval variables at each hierarchy level and introduce the related constraints.

Overview of the Hierarchical Structure

Figure 4-1 gives a brief summary of the interval variable hierarchy. Each level has its own purpose, and the relationship between any two levels is established by CP constraints.

At the top level, we define the job interval variable to record the start/end time of the job. Therefore, we can easily calculate the makespan by comparing the end time of each job and finding the latest one (refer to Fig. 4-2 on page 62 for better understanding).

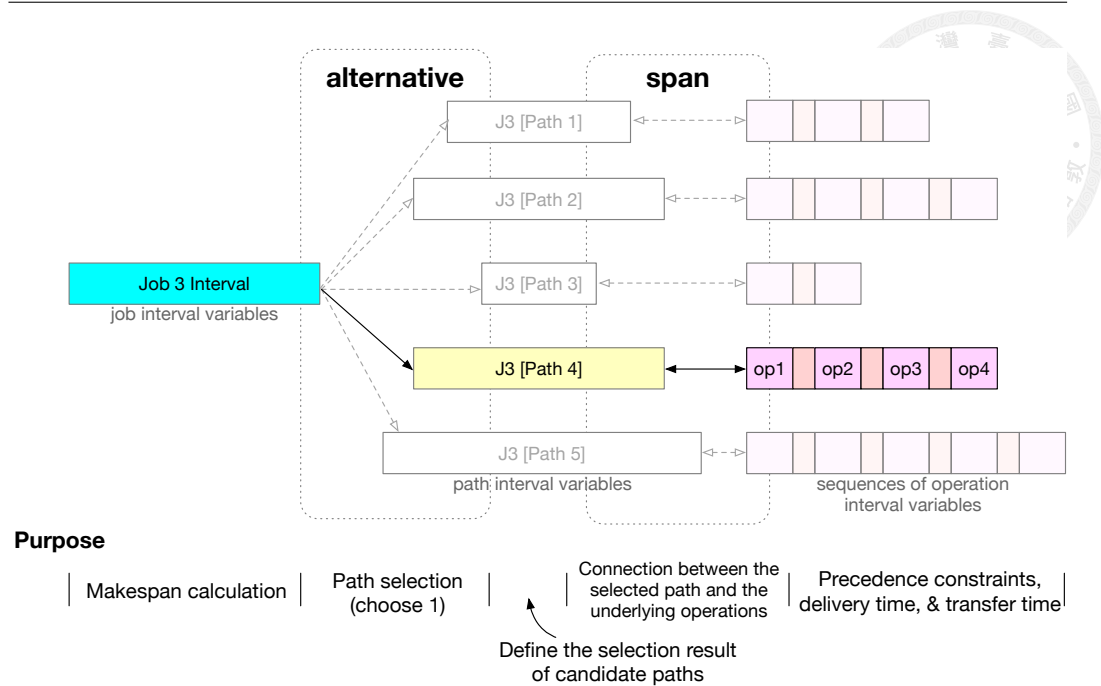
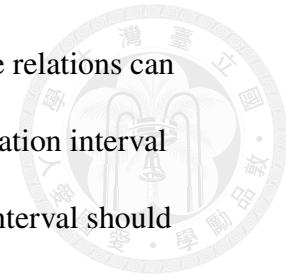


Figure 4-1. Illustration of the hierarchical structure of the interval variables.

At the middle level, we define the path interval variables. In CP, an interval variable can be specified as “optional,” meaning that the solver would determine whether the interval participates in the solution or not. We set all path interval variables to be optional and use the constraint $alternative(x, X)$ to mandate that *one and only one* of the paths (in set X) should be “present” in the solution for each job (variable x). Moreover, the start/end time of the job would align with those of the selected path by this constraint as well. This step is equivalent to the path selection in solving an MHNSP.

At the lower level, operation interval variables are defined. Again, we set all operation interval variables to be optional. When a path is selected, i.e., present, the underlying operation interval variables should also be present. At the same time, all the other non-selected paths and their associated operations should be “absent”. Moreover, the start/end time of a path should align with the start time of the first



operation and the end time of the last one when present. All the above relations can be achieved by the constraints $span(x, X)$, where X is the set of operation interval variables and x is the path interval variable. Finally, each operation interval should respect the precedence constraints, namely, they are executed sequentially.

Without this hierarchical structure, it would be hard to model the relationship between jobs, candidate paths, and operations elegantly. As shown in Fig. 4-2, several jobs with numerous candidate paths and operations are to be scheduled and solved by the CP model. It will be extremely cumbersome to define the relationship “manually”, which will be the case in our Integer Programming model.

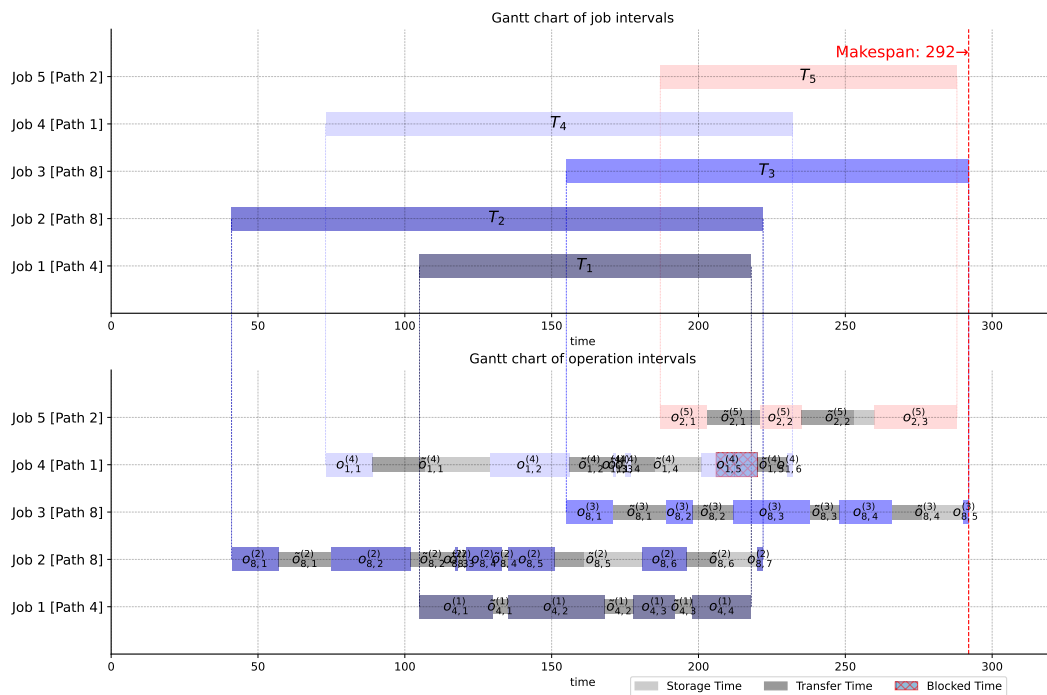


Figure 4-2. Gantt chart results of a CP solution.

Next, we will formally define the interval variables in each level and construct their associated constraints.

Job Interval Variable, Makespan Calculation, and Optimization Goal

At the top level of the hierarchy, we define the time interval variable T_k for job J_k , representing the delivery time period in the final schedule of the solution to the MHNSP. Let $\mathbf{T} = \{T_k | k = 1, 2, \dots, n\}$ be the interval variable set for all jobs. The makespan is then derived from the job intervals as

$$C^{\max} = \max_{\forall T_k \in \mathbf{T}} \text{endOf}(T_k), \quad (4.1)$$

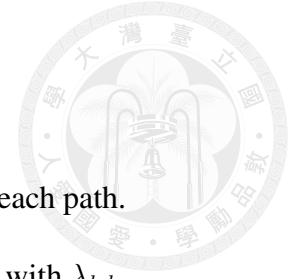
where $\text{endOf}(T_k)$ returns the end time of job J_k . Figure 4-2 on the previous page indicates that the makespan is calculated by the latest end time of all jobs in the scheduled solution.

Candidate Path Interval Variable

Following the proposed modeling hierarchy, we define candidate path interval variables for a job. Notice that there are σ_k paths in candidate set Π_k for job J_k . We define σ_k “optional” interval variables for the job. Let $t_l^{(k)}$ be the interval variable for the l -th candidate path $\pi_l^{(k)}$, $l = 1, 2, \dots, \sigma_k$. Assume that the set of path interval variables for the job J_k is $C_k = \{t_1^{(k)}, t_2^{(k)}, \dots, t_{\sigma_k}^{(k)}\}$.

In CP modeling, selecting one optional interval from a candidate set for a target interval is achieved via the constraining operator $\text{alternative}(x, X)$, where X is the candidate interval set for the target interval x . Therefore, to facilitate the path selection, we have constraints

$$\text{alternative}(T_k, C_k), \quad k = 1, 2, \dots, n.$$



Operation Interval Variable

Now, we define the lowest-level operation interval variables of each path.

Let $\pi_l^{(k)} = \langle \rho_{l,1}^{(k)}, \rho_{l,2}^{(k)}, \dots, \rho_{l,\lambda_{k,l}}^{(k)} \rangle$ be the l -th candidate path of job J_k with $\lambda_{k,l}$ sequential delivery operations. Note that a transfer operation exists between a pair of consecutive delivery operations in the path. Let the sequence of the transfer sites visited be $\langle \epsilon_{l,1}^{(k)}, \epsilon_{l,2}^{(k)}, \dots, \epsilon_{l,\lambda_{k,l}-1}^{(k)} \rangle$, where the transfer site $\epsilon_{l,j}^{(k)}$ conducts a transfer operation between operations $\rho_{l,j}^{(k)}$ and $\rho_{l,j+1}^{(k)}$.

To construct our CP model, we define the corresponding set of interval variables $O_l^{(k)} = \{o_{l,1}^{(k)}, o_{l,2}^{(k)}, \dots, o_{l,\lambda_{k,l}}^{(k)}\}$ for the “delivery operations” of the l -th candidate path. Normally, the CP solver determines the interval’s start time, which is subject to vehicle availability in the node. Between two consecutive delivery operations, there is a transfer operation conducted in the transfer site whose start time is to be determined by the solver against the buffer size constraint. We have $\lambda_{k,l} - 1$ transfer time intervals to be defined and determined by the solver. Let $\tilde{O}_l^{(k)} = \{\tilde{o}_{l,1}^{(k)}, \tilde{o}_{l,2}^{(k)}, \dots, \tilde{o}_{l,\lambda_{k,l}-1}^{(k)}\}$ be the defined interval variable set for the “transfer operations.” Note that the mapped transfer site for interval $\tilde{o}_{l,j}^{(k)}$ is $\epsilon_{l,j}^{(k)}$. In Fig. 4-2 on page 62, the delivery operations are represented by the “thicker” blocks, while the thinner blocks are the transfer operations.

Since each delivery operation interval $o_{l,1}^{(k)}$ represents the time span of the material load in the node and sometimes the load would be blocked, it should therefore be longer than the delivery time required (since the vehicle is still

occupied by the material load, the operation is not completed). That is

$$\text{lengthOf}(o_{l,j}^{(k)}) \geq \tau_{\theta,\delta}^{(\eta)}, \rho_{l,j}^{(k)} \equiv (\eta, \theta, \delta),$$

$$j = 1, 2, \dots, \lambda_{k,l}, l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n.$$

Similarly, each transfer operation interval $\tilde{o}_{l,1}^{(k)}$ should be longer than the transfer time required since the material might not be able to be promptly picked up by the next node after the transfer operation. We have

$$\text{lengthOf}(\tilde{o}_{l,j}^{(k)}) \geq \beta_d, \text{ where } \epsilon_{l,j}^{(k)} \equiv e_d,$$

$$j = 1, 2, \dots, \lambda_{k,l} - 1, l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n.$$

Since a path for a job is a sequence of operations executed consecutively, the CP model has precedence constraints applied to operation intervals in determining their start times. In CP modeling, the constraining operator $\text{endAtStart}(x, x')$ restricts the start time of interval x' to be equal to the end time of x when both intervals are present; i.e.,

$$\left(\text{presenceOf}(x) \wedge \text{presenceOf}(x') \right) \Rightarrow \left(\text{endOf}(x) = \text{startOf}(x') \right),$$

where \wedge is the AND Boolean operator. Therefore, a precedence constraint is applied to a consecutive pair of delivery operation interval and transfer operation interval, i.e., $o_{l,j}^{(k)}, \tilde{o}_{l,j}^{(k)}$ and $\tilde{o}_{l,j}^{(k)}, o_{l,j+1}^{(k)}$.

We know the CP solver will select one candidate path for each job. If the l' -th path is selected for job J_k , the interval variable $t_{l'}^{(k)}$ will be present in the solution while others are absent. In addition, the start and end times of the job interval variable T_k should align with those of $t_{l'}^{(k)}$. This can be achieved by the



$span(\cdot, \cdot)$ constraint, namely

$$span(t_l^{(k)}, O_l^{(k)}), \quad l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n$$

Figure 4-3 illustrates the associated relationship among the job interval variable, the selected path interval variable, and the list of operation interval variables of the path. In the figure, the 40th candidate path is selected by the solver for job J_3 . The CP solver will set

$$\begin{aligned} startOf(T_3) &\leftarrow startOf(t_{40}^{(3)}) \leftarrow startOf(o_{40,1}^{(3)}), \\ endOf(T_3) &\leftarrow endOf(t_{40}^{(3)}) \leftarrow endOf(o_{40,4}^{(3)}). \end{aligned}$$

Note that the selected path has four delivery and three transfer intervals defined.

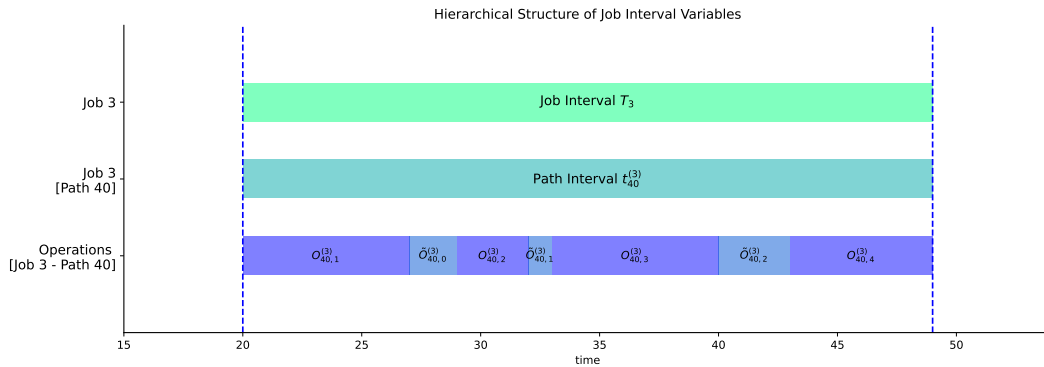


Figure 4-3. Alignment Relationship between job interval variable, path interval variable, and operation interval variable.

Summary

The CP model relating to the constraints of jobs is summarized as follows.

Definitions of Interval Variables

$$\mathbf{T} = \{T_k \mid k = 1, 2, \dots, n\} \tag{4.2}$$

$$C_k = \{t_l^{(k)} \mid l = 1, 2, \dots, \sigma_k\}, \quad k = 1, 2, \dots, n \tag{4.3}$$

$$O_l^{(k)} = \{o_{l,j}^{(k)} \mid j = 1, 2, \dots, \lambda_{k,l}\}, \quad l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n \quad (4.4)$$

$$\tilde{O}_l^{(k)} = \{\tilde{o}_{l,j}^{(k)} \mid j = 1, 2, \dots, \lambda_{k,l} - 1\}, \quad l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n \quad (4.5)$$

Constraints:

$$\text{alternative}(T_k, C_k), \quad k = 1, 2, \dots, n \quad (4.6)$$

$$\text{span}(t_l^{(k)}, O_l^{(k)}), \quad l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n \quad (4.7)$$

$$\text{presenceOf}(t_l^{(k)}) \iff \text{presenceOf}(o_{l,j}^{(k)}) \quad (4.8)$$

$$j = 1, 2, \dots, \lambda_{k,l}, l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n$$

$$\text{presenceOf}(t_l^{(k)}) \iff \text{presenceOf}(\tilde{o}_{l,j}^{(k)}), \quad (4.9)$$

$$j = 1, 2, \dots, \lambda_{k,l} - 1, l = 1, 2, \dots, \sigma_k,$$

$$k = 1, 2, \dots, n$$

$$\text{endAtStart}(o_{i,j}^{(k)}, \tilde{o}_{i,j}^{(k)}), \quad j = 1, 2, \dots, \lambda_{k,l} - 1, \quad l = 1, 2, \dots, \sigma_k, \quad (4.10)$$

$$k = 1, 2, \dots, n$$

$$\text{endAtStart}(\tilde{o}_{i,j}^{(k)}, o_{i,j+1}^{(k)}), \quad j = 1, 2, \dots, \lambda_{k,l} - 1, \quad l = 1, 2, \dots, \sigma_k, \quad (4.11)$$

$$k = 1, 2, \dots, n$$

$$\text{presenceOf}(o_{l,j}^{(k)}) \implies (\text{lengthOf}(o_{l,j}^{(k)}) \geq \tau_{\theta,\delta}^{(\eta)}),$$

$$\text{where } \rho_{l,j}^{(k)} \equiv (\eta, \theta, \delta), j = 1, 2, \dots, \lambda_{k,l}, l = 1, 2, \dots, \sigma_k,$$

$$k = 1, 2, \dots, n$$

$$(4.12)$$



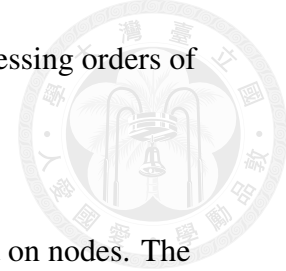
$$\begin{aligned}
 \text{presenceOf}(\tilde{o}_{l,j}^{(k)}) &\implies (\text{lengthOf}(\tilde{o}_{l,j}^{(k)}) \geq \beta_d) \\
 \text{where } \epsilon_{l,j}^{(k)} &\equiv e_d, j = 1, 2, \dots, \lambda_{k,l} - 1, l = 1, 2, \dots, \sigma_k, \\
 k &= 1, 2, \dots, n
 \end{aligned}
 \tag{4.13}$$

Equations 4.2 to 4.5 define the job interval variables, path interval variables, and operation interval variables. Equation 4.6 restricts that each job should select one and only one path. Equation 4.7 further imposes the alignment of start/end time on the operation interval variables and their corresponding path interval variable. Moreover, the presence of the operation interval variables in the same path should be the same, as described in Eqs. 4.8 and 4.9. The operations should also comply with the precedence constraints and be executed sequentially. This constraint is imposed by Eq. 4.10 and 4.11. Finally, the length of each operation should be equal to or greater than the given delivery time or transfer time, which are described in Eq. 4.12 and 4.13.

In this subsection, we have constructed part of the CP model that is related to the transportation job set. In the following subsections, we will investigate the material handling operations on a node and a transfer site whose interval variables have been introduced for jobs.

4.1.2 Operation Constraints on Nodes

In the MHNSP, the vehicle of a node receives a set of operations decomposed from candidate paths of all jobs. The interval variables for these delivery operations acquire the transportation resources to move from the pick-up



point to the drop-off point. The CP solver will arrange the processing orders of these operations on each node to achieve the optimization goal.

Figure 4-4 shows an example of the operations scheduled on nodes. The solution is the same as the one shown in Fig. 4-2 in the previous subsection, except that we reorganize the operation intervals from the nodes' perspective. As shown in the figure, the delivery operations in a node are aggregated from different jobs and different paths. For example, the operations on node 8 are from job J_1 , J_2 , J_3 , and J_4 . Therefore, the main idea in constructing the constraints in nodes is to gather all the operation interval variables executed on each node in a set and establish the constraints of empty-car moving time between two consecutive operations when they are both present.

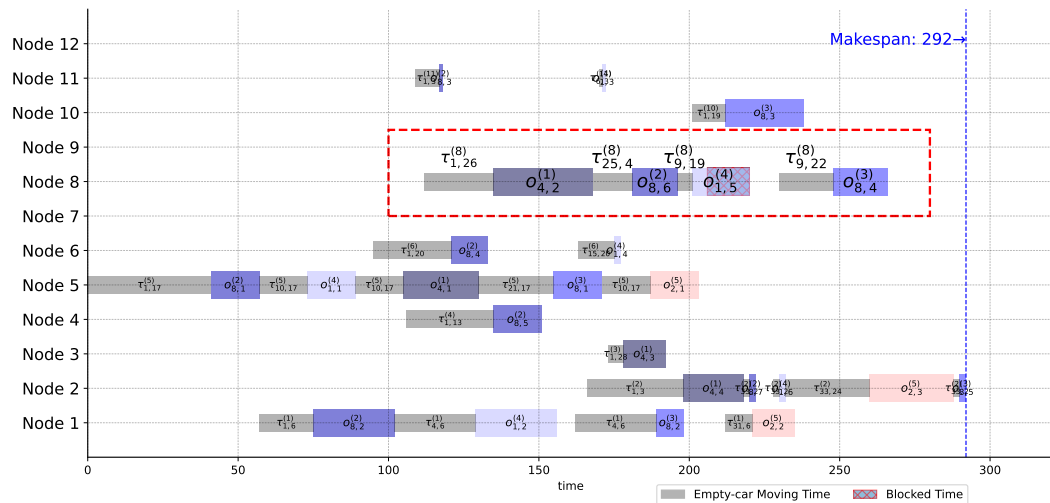


Figure 4-4. Gantt chart result of nodes from the CP model.

In CP modeling, the constraining operator $noOverlap(Q)$ restricts the intervals sequentialized by the sequence variable Q not to overlap each other. We need to collect operation interval variables that are to be executed on each transportation node to model the processing constraints on the node. Let the



delivery operation interval set for node i be

$$X^{(i)} = \left\{ o_{l,j}^{(k)} \mid \rho_{l,j}^{(k)} \equiv (\eta_{l,j}^{(k)}, \theta_{l,j}^{(k)}, \delta_{l,j}^{(k)}), \eta_{l,j}^{(k)} = i; \right. \\ \left. j = 1, 2, \dots, \lambda_{k,l}, l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n \right\}.$$

Note that operations are from different paths of different jobs yet executed on node

i . In our MHNSP, the delivery operation $\rho \equiv (i, \theta, \delta)$ represented by an interval variable $o \in X^{(i)}$ is the actual material movement on node i . If $\rho' \equiv (i, \theta', \delta')$ is the next operation of ρ and its interval variable is o' , an empty-car moving time $\tau_{\delta, \theta'}^{(i)}$ is required between intervals o and o' , which is the transition time between two successive intervals (operations).

In CP, the transition time between two successive intervals in a sequence variable can be modeled by setting a positive “type” integer to each interval in the sequence. The transition time from one type to another can be specified in advance. Since a delivery operation is executed from a pick-up point to a drop-off point, there are κ_i^2 “types” of operations. Follow Liu and Yang[16], suppose we define the type of an operation interval as the product of its pick-up index and the number of P/D points of the node plus the drop-off point index. Therefore, the type of interval o representing operation $\rho \equiv (i, \theta, \delta)$ is $y = (\theta - 1) \cdot \kappa_i + \delta$, and the type of o' is $y' = (\theta' - 1) \cdot \kappa_i + \delta'$. We can set the transition time from an interval of type y to an interval of type y' as the empty-car time $\tau_{\delta, \theta'}^{(i)}$. We can ask the CP solver to separate two non-overlapped intervals with the specified transition time for us. In our CP model, the transition times are the empty-car times between two delivery operations on the node.

In CP modeling, the interval variables specified in an “advanced sequence

variable” must have an integer type specified. The enhanced definition of the sequence variable is $Q \equiv \text{seq}(X, Y)$, where Y is the associated list of types of the intervals in X . Associated with this enhancement, the constraining operator $\text{noOverlap}(Q, \Phi)$ separates non-overlapped intervals with transition times specified in the matrix $\Phi = [\phi_{j,j'}]$. The transition time $\phi_{j,j'} \geq 0$ is used to separate an interval variable of type j' from its preceding interval of type j . Therefore, in the solution of our example, $\text{endOf}(o) + \phi_{j,j'} \leq \text{startOf}(o')$.

To take advantage of the enhancement, we define sequence variables for nodes as $Q^{(i)} \equiv \text{seq}(X^{(i)}, Y^{(i)})$, $i \in V$. The integer type list

$$Y^{(i)} \equiv \left[\left(\theta_{l,j}^{(k)} - 1 \right) \cdot \kappa_i + \delta_{l,j}^{(k)} \mid \eta_{l,j}^{(k)} = i; \right. \\ \left. j = 1, 2, \dots, \lambda_{k,l}, l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n \right]$$

is the associated type values of the intervals $o_{l,j}^{(k)}$ in $X^{(i)}$. Note that the dimension of the transit time matrix for node i is $\kappa_i^2 \times \kappa_i^2$. The transit time matrix is $\Phi^{(i)} = [\phi_{j,j'}^{(i)}]_{\kappa_i^2 \times \kappa_i^2} \equiv [\tau_{\theta^*, \delta^*}^{(i)}]_{\kappa_i^2 \times \kappa_i^2}$, where $\theta^* = (j - 1) \bmod \kappa_i + 1$ and $\delta^* = \lceil \frac{j'}{\kappa_i} \rceil$. Then, the enhanced constraints used in our CP model is $\text{noOverlap}(Q^{(i)}, \Phi^{(i)})$, $\forall i \in V$.

For example, if node i has three consecutive operations $\rho_1 \equiv (i, 2, 1)$, $\rho_2 \equiv (i, 2, 3)$, and $\rho_3 \equiv (i, 1, 3)$ to be executed. The interval set $X^{(i)} = \{x_1, x_2, x_3\}$ is defined for these three operations. Suppose node i has $\kappa_i = 3$ P/D points with from-to time matrix

$$M^{(i)} = \begin{bmatrix} 0 & 1 & 2 \\ 4 & 0 & 3 \\ 5 & 6 & 0 \end{bmatrix}$$

Since there are $\kappa_i \cdot \kappa_i$ combinations for a pick-up/drop-off pair, $\kappa_i^2 = 9$ types



of delivery operations exist. Therefore, the transit time matrix from type to type is

$$\Phi^{(i)} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 4 & 4 & 4 & 0 & 0 & 0 & 3 & 3 & 3 \\ 5 & 5 & 5 & 6 & 6 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 4 & 4 & 4 & 0 & 0 & 0 & 3 & 3 & 3 \\ 5 & 5 & 5 & 6 & 6 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 4 & 4 & 4 & 0 & 0 & 0 & 3 & 3 & 3 \\ 5 & 5 & 5 & 6 & 6 & 6 & 0 & 0 & 0 \end{bmatrix}$$

From the type definition, the type of interval x_1 is $y_1 = (2 - 1) \cdot 3 + 1 = 4$. Similarly, $y_2 = (2 - 1) \cdot 3 + 3 = 6$ and $y_3 = (1 - 1) \cdot 3 + 3 = 3$. In the CP model, the sequence variable for node i is then $Q^{(i)} \equiv seq(\{x_1, x_2, x_3\}, [4, 6, 3])$. Suppose the enhanced constraint $noOverlap(Q^{(i)}, \Phi^{(i)})$ is specified and $x_2 \mapsto x_3 \mapsto x_1$ is a value for the sequence variable $Q^{(i)}$. The corresponding type sequence $y_2 \mapsto y_3 \mapsto y_1$ is $6 \mapsto 3 \mapsto 4$. Then empty-car time $\phi_{6,3}^{(i)} = 5 = \tau_{3,1}^{(i)}$ separates operations $\rho_2 \equiv (i, 2, 3)$ and $\rho_3 \equiv (i, 1, 3)$, where the vehicle travels empty from point 3 to 1. Therefore, in the solution

$$endOf(x_2) + \phi_{6,3}^{(i)} \leq startOf(x_3).$$

Similarly, $\phi_{3,4}^{(i)} = 6 = \tau_{3,2}^{(i)}$ separates delivery operations $\rho_3 \equiv (i, 1, 3)$ and $\rho_1 \equiv (i, 2, 1)$ for traveling from points 3 to 2.

Notice that all node vehicles initially reside at the 1st P/D point. To model the empty-car moving time to the pick-up point of the first operation executed by a node, a dummy interval variable is added to the interval set. That is $X^{(i)} \leftarrow X^{(i)} \cup \{o_0^{(i)}\}$, where $startOf(o_0^{(i)}) = 0$ and $lengthOf(o_0^{(i)}) = 0$. The type of dummy interval is $y_0^{(i)} = 1$, standing for a dummy operation that starts and ends at point 1. The type list is updated as $Y^{(i)} \leftarrow Y^{(i)} \oplus y_0^{(i)}$. Since the start time of the



dummy interval is set to 0, it will always be the first interval appearing in the solution of a sequence variable.

Summary

The CP Model relating to constraints on nodes is summarized as follows:

Definitions of Interval and Sequence Variables

$$Q^{(i)} \equiv seq(X^{(i)}, Y^{(i)}), \forall i \in V, \quad (4.14)$$

where

$$X^{(i)} = \left\{ o_{l,j}^{(k)} \mid \rho_{l,j}^{(k)} \equiv (\eta_{l,j}^{(k)}, \theta_{l,j}^{(k)}, \delta_{l,j}^{(k)}), \eta_{l,j}^{(k)} = i; j = 1, 2, \dots, \lambda_{k,l}, \right. \\ \left. l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n \right\} \cup \{o_0^{(i)}\} \quad (4.15)$$

$$Y^{(i)} \equiv \left[(\theta_{l,j}^{(k)} - 1) \cdot \kappa_i + \delta_{l,j}^{(k)} \mid \eta_{l,j}^{(k)} = i; j = 1, 2, \dots, \lambda_{k,l}, \right. \\ \left. l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n \right] \oplus 1 \quad (4.16)$$

Constraints

$$startOf(o_0^{(i)}) = 0, \forall i \in V \quad (4.17)$$

$$lengthOf(o_0^{(i)}) = 0, \forall i \in V \quad (4.18)$$

$$noOverlap(Q^{(i)}, \Phi^{(i)}), \forall i \in V, \quad (4.19)$$

where $\Phi^{(i)} = [\phi_{j,j'}^{(i)}]_{\kappa_i^2 \times \kappa_i^2} \equiv [\tau_{\theta^*, \delta^*}^{(i)}]_{\kappa_i^2 \times \kappa_i^2}$, where $\theta^* = (j - 1) \bmod \kappa_i + 1$ and $\delta^* = \lceil \frac{j'}{\kappa_i} \rceil$.

Equation 4.14 defines the sequence variable for each node, where $X^{(i)}$ is the set of intervals for delivery operations executed on node i , and $Y^{(i)}$ is the list of associated types for $X^{(i)}$ to model the empty-car moving time between two

consecutive operations. Equations 4.17 and 4.18 restrict the dummy interval $o_0^{(i)}$ to have zero length and start at time 0. Finally, Equation 4.19 states that there can be at most one operation executing at any moment on each node, and there is an empty-car moving time between two consecutive operations.

In the next subsection, we will describe the techniques for modeling a transfer site having capacity limit and transfer time requirement.

4.1.3 Operation Constraints on Transfer Sites

A transfer operation of a job is conducted at a transfer site right after a delivery operation is executed by a node vehicle. However, the vehicle is blocked if there is no vacancy on the transfer site. A “blocked” vehicle cannot drop the load at the drop-off point, which is also the source point of the successive transfer site. When a vacancy is available, the vehicle detaches the load and is freed for the next delivery operation. On the other hand, the once-blocked job starts the transfer operation at the transfer site until it reaches the target point.

The number of transfer operations conducted simultaneously in the transfer site e_d is limited by its buffer size ω_d , and each transfer takes at least β_d time to reach the target point. Therefore, buffer size restricts the operation count on the transfer site. Since the transfer time in the transfer site e_d is β_d , a transfer interval variable x contributes to the count function of the site by 1 from $startOf(x)$ to $endOf(x)$. This contribution can be regarded as a pulse function whose value is 1 in the range of the interval and 0 outside the interval. Adding up all the pulse functions on the transfer site e_d yields a cumulative count function. Figure 4-5

shows a cumulative count function that adds up 5 pulse functions from interval variables whose lengths are generally different.

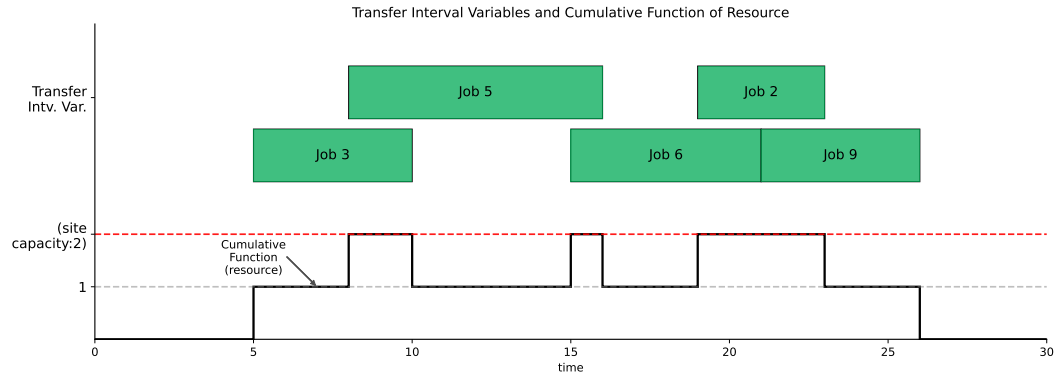


Figure 4-5. Illustration of transfer operation interval variables and cumulative function of resource.

In modeling the buffer size constraints on transfer sites, we first aggregate the transfer operation interval variables for each transfer site. The interval sets for all transfer sites are

$$\Gamma^{(d)} = \left\{ \tilde{\delta}_{l,j}^{(k)} \mid \epsilon_{l,j}^{(k)} = e_d; j = 1, 2, \dots, \lambda_k - 1, k = 1, 2, \dots, n; \right. \\ \left. l = 1, 2, \dots, \sigma_k \right\}, d = 1, 2, \dots, r.$$

Note that $\epsilon_{l,j}^{(k)}$ is the transfer site that conducts transfer operation between delivery operations $\rho_{l,j}^{(k)}$ and $\rho_{l,j+1}^{(k)}$, where the transfer operation is represented by interval $\tilde{\delta}_{l,j}^{(k)}$. The cumulative count function of site e_d is

$$C^{(d)} \equiv \sum_{o \in \Gamma^{(d)}} pulse(o, 1).$$

The constraint of buffer size is then modeled as

$$alwaysIn(C^{(d)}, o, 0, \omega_d), \forall o \in \Gamma^{(d)}, d = 1, 2, \dots, r.$$



Summary

The CP Model relating to constraints on transfer sites is summarized as follows.

Definitions of Interval Sets

$$\Gamma^{(d)} = \left\{ \tilde{o}_{l,j}^{(k)} \mid \epsilon_{l,j}^{(k)} = e_d; j = 1, 2, \dots, \lambda_k - 1, k = 1, 2, \dots, n; \right. \\ \left. l = 1, 2, \dots, \sigma_k \right\}, d = 1, 2, \dots, r. \quad (4.20)$$

Constraints

$$alwaysIn(C^{(d)}, o, 0, \omega_d), \forall o \in \Gamma^{(d)}, d = 1, 2, \dots, r \quad (4.21)$$

where $C^{(d)} \equiv \sum_{\forall o \in \Gamma^{(d)}} pulse(o, 1)$.

The capacity limit of each site is imposed in Eq. 4.21, where ω_d is the capacity limit for site d .

4.2 Integer Programming Model for Material Handling Network Scheduling Problem

This section presents the Integer Programming Model for solving the MHNSP. Following the structure of the proposed CP model in the previous section, we first introduce the decision variables of the model and depict the optimization objective. After that, detailed constraint construction steps for transportation jobs, nodes, and transfer sites are presented. The buffer size constraints on transfer sites, which incur complex logical formulations, are last deliberated.

The main difference from the proposed CP model is that we do not use the hierarchical structure for the IP model. In fact, the IP model only solves for the *start time of each operation* as shown in Fig. 4-6. Note that the solution is the same as the example presented in Fig. 4-2 on page 62 in the previous section.

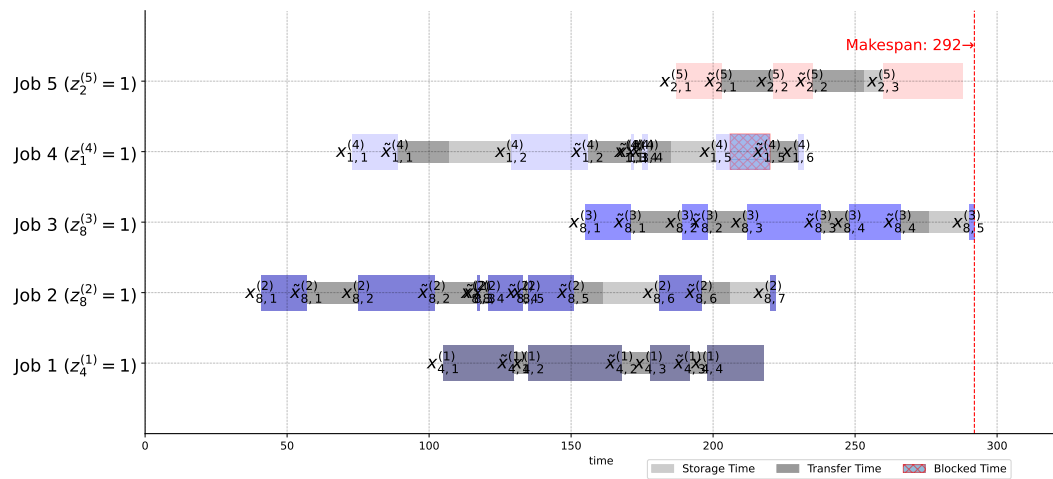


Figure 4-6. Gantt chart results of an IP solution.

4.2.1 Decision Variables and Optimization Goal

Solving an MHNSP involves selecting a candidate path for each job and scheduling the start time for each operation on the path, including delivery and transfer operations. The goal is to minimize the longest makespan (completion time) among the scheduled jobs. In the following, we will introduce the decision variables of the proposed integer programming model, where the makespan and the optimization goal can be expressed in terms of these variables.



Decision Variables

Path selection and *operation sequencing* are two primary decision tasks in solving an MHNSP. The path selection uses a binary variable for each candidate path to signal whether the path is selected. The binary variable is $z_l^{(k)} \in \{0, 1\}$, $l = 1, 2, \dots, \sigma_k$, $k = 1, 2, \dots, n$, where $z_l^{(k)} = 1$ indicates the l -th candidate path is selected for job J_k . Notice that precisely one path is selected in a valid solution. The operation sequencing results are illustrated by setting the delivery and transfer start times for *delivery* operations on nodes and *transfer* operations on transfer sites, respectively. Let the variable

$x_{l,j}^{(k)}$, $j = 1, 2, \dots, \lambda_{l,k}$, $l = 1, 2, \dots, \sigma_k$, $k = 1, 2, \dots, n$ represent the start time of the j -th *delivery operation* on the l -th candidate path for job J_k . Let the variable $\tilde{x}_{l,j}^{(k)}$, $j = 1, 2, \dots, \lambda_{l,k} - 1$, $l = 1, 2, \dots, \sigma_k$, $k = 1, 2, \dots, n$ represent the start time of the j -th *transfer operation* on the l th candidate path for job J_k . Note that the transfer operation with start time $\tilde{x}_{l,j}^{(k)}$ is conducted on transfer site $\epsilon_{l,j}^{(k)}$.

Optimization Goal

The objective of solving the MHNSP is to minimize the maximal makespan of jobs. We define a decision variable for the makespan C^{\max} . Therefore, the goal is:

$$\min C^{\max}, \tag{4.22}$$

subject to the following part of constraints:

$$x_{l,\lambda_{k,l}}^{(k)} + \tau_{\tilde{\theta}, \tilde{\delta}}^{(\tilde{\eta})} \leq L \left(1 - z_l^{(k)}\right) + C^{\max}, \rho_{l,\lambda_{k,l}}^{(k)} \equiv (\tilde{\eta}, \tilde{\theta}, \tilde{\delta}), \tag{4.23}$$

$$l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n,$$

where L is a large number. Here, $\tilde{\eta}$, $\tilde{\theta}$, and $\tilde{\delta}$ denote the node, pick-up point index, and drop-off index of the last delivery operation on the l -th candidate path for job J_k . The constraint ensures that for any selected path, the makespan C^{\max} must be larger than or equal to the completion time of the last delivery operation on that path.

4.2.2 Variable Constraints on Each Job

This subsection illustrates the constraints of the introduced variables on the transportation jobs. Two primary constraints on each job are: only one candidate path is selected, and the operation execution precedences on each candidate path are kept. To ensure that only one path is selected,

$$\sum_{l=1}^{\sigma_k} z_l^{(k)} = 1, \quad k = 1, 2, \dots, n. \quad (4.24)$$

For the precedence constraint, the end time of a delivery operation must be before the successive transfer operation if the path is selected. Therefore,

$$x_{l,j}^{(k)} + \tau_{\tilde{\theta}, \tilde{\delta}}^{(\tilde{\eta})} \leq \tilde{x}_{l,j}^{(k)} + L(1 - z_l^{(k)}), \quad j = 1, 2, \dots, \lambda_{l,k} - 1, \quad (4.25)$$

$$l = 1, 2, \dots, \sigma_k, \quad k = 1, 2, \dots, n,$$

where $\rho_{l,j}^{(k)} \equiv (\tilde{\eta}, \tilde{\theta}, \tilde{\delta})$. Similarly, the end time of a transfer operation must be before the next delivery operation,

$$\tilde{x}_{l,j}^{(k)} + \beta_d \leq x_{l,j+1}^{(k)} + L(1 - z_l^{(k)}), \quad j = 1, 2, \dots, \lambda_{l,k} - 1, \quad (4.26)$$

$$l = 1, 2, \dots, \sigma_k, \quad k = 1, 2, \dots, n,$$

where $\epsilon_{l,j}^{(k)} \equiv e_d$ —the transfer site that conducts the transfer operation associated with $\tilde{x}_{l,j}^{(k)}$ is e_d .

In these constraints, L is a given large number. The equation ensures that if



the l -th optional path is selected for job J_k , all operations within this path must follow the delivery orders.

4.2.3 Operation Constraints on Nodes

The operation constraints on nodes focus on the required empty-car moving times between two consecutive delivery operations. Similar to the development of the CP model, we need to construct the variable set representing the delivery operations executed on each node

$$X^{(i)} = \left\{ x_{l,j}^{(k)} \mid \rho_{l,j}^{(k)} \equiv (\eta, \theta, \delta), \eta = i; \right. \\ \left. j = 1, 2, \dots, \lambda_{l,k}, l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n \right\}, \forall i \in V.$$

Since the operations executed on node i are yielded from different paths of different jobs and their execution order are to be determined, we need to reorganize the variables in $X^{(i)}$ for modeling convenience. We collect the operation start time variables with their job indexes, path indexes, pick-up, and drop-off point indexes as 5-tuples and index the tuples from 1 to the total count of delivery operations on node i . Therefore, if the index for variable $x_{l,j'}^{(k)}$ is j , whose operation is

$$\rho_{l,j'}^{(k)} \equiv (i, \theta_{l,j'}^{(k)}, \delta_{l,j'}^{(k)}), \text{ the tuple is } (y_{j'}^{(i)}, \hat{k}_{j'}^{(i)}, \hat{l}_{j'}^{(i)}, \hat{\theta}_{j'}^{(i)}, \hat{\delta}_{j'}^{(i)}) \equiv (x_{l,j'}^{(k)}, k, l, \theta_{l,j'}^{(k)}, \delta_{l,j'}^{(k)})$$

Let the set of the indexed tuples mapped from $X^{(i)}$ be

$$Y^{(i)} = \left\{ (y_j^{(i)}, \hat{k}_j^{(i)}, \hat{l}_j^{(i)}, \hat{\theta}_j^{(i)}, \hat{\delta}_j^{(i)}) \mid j = 1, 2, \dots, \gamma_i \right\}, \forall i \in V,$$

where $\gamma_i = |X^{(i)}|$ is the number of delivery operations executed on node i . The integer programming model will sequentialize the operation orders on each node.

Since there is only one vehicle available for each node, the start time of the successive operation must be later than the end time of the preceding operation

plus the empty-car moving time. In Fig. 4-7, the delivery operation $\rho_{1,1}^{(4)}$ with the start time $x_{1,1}^{(4)}$ is executed right after $\rho_{8,1}^{(2)}$ with the start time $x_{8,1}^{(2)}$ on node 5; therefore, there is an empty-car moving time $\tau_{10,17}^{(5)}$ from P/D point $p_{10}^{(5)}$ to $p_{17}^{(5)}$.

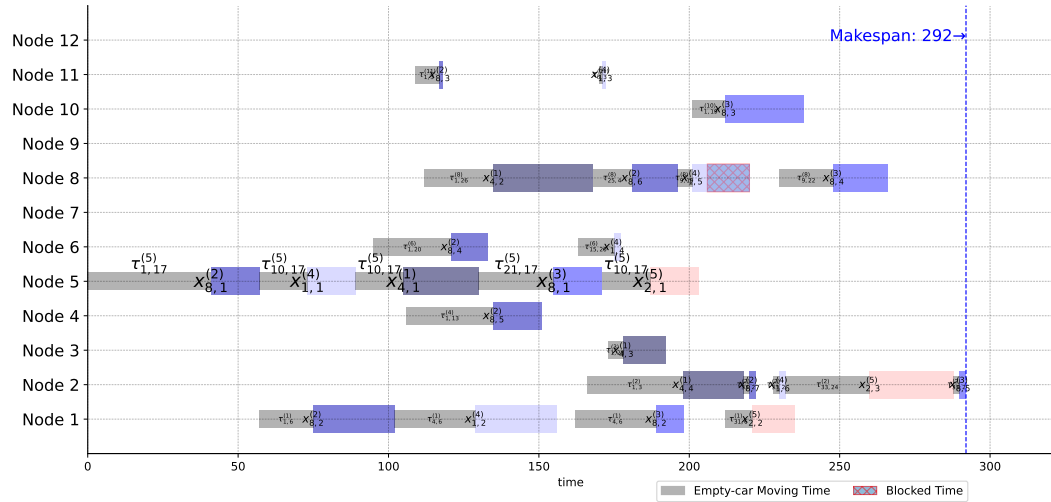


Figure 4-7. Gantt chart results of an IP solution on nodes.

If the operation of the j -th tuple is executed right after the j' -th tuple, then

$$y_{j'}^{(i)} + \tau_{\hat{\theta}_{j'}^{(i)}, \hat{\delta}_{j'}^{(i)}} + \tau_{\hat{\delta}_{j'}^{(i)}, \hat{\theta}_j^{(i)}} \leq y_j^{(i)}. \text{ In the above example, it means}$$

$$x_{8,1}^{(2)} + \tau_{17,10}^{(5)} + \tau_{10,17}^{(5)} \leq x_{1,1}^{(4)}. \text{ To set the succeeding/preceding relationship, we need}$$

the binary variables for the operations of j' -th and j -th tuples. We define the binary

variable $q_{j',j}^{(i)} \in \{0, 1\}$, $\forall j', j \in \{1, 2, \dots, \gamma_i\}, j' \neq j$. $q_{j',j}^{(i)} = 1$ indicates that the operation of the j -th tuple is executed right after that of the j' -th. These binary

variables help to establish an operation sequence on each node. Since there are γ_i operations executed in node i , there will be $\gamma_i - 1$ variables with value 1 assigned while others are 0. However, there are some circumstances where no operation is

“actually” executed in node i . From the perspective of CP, this means that all the

operation interval variables on the node are “absent”. We introduce the binary

variable $\bar{q}^{(i)} \in \{0, 1\}$ for node i . When $\bar{q}^{(i)} = 1$, there is more than one operation



present in node i . The variables introduced and constraints on these variables are listed below.

Summary

Definitions of Variable sets

$$Y^{(i)} = \left\{ \left(y_j^{(i)}, \hat{k}_j^{(i)}, \hat{l}_j^{(i)}, \hat{\theta}_j^{(i)}, \hat{\delta}_j^{(i)} \right) \mid j = 1, 2, \dots, \gamma_i \right\}, \forall i \in V, \quad (4.27)$$

where the j -th tuple $\left(y_j^{(i)}, \hat{k}_j^{(i)}, \hat{l}_j^{(i)}, \hat{\theta}_j^{(i)}, \hat{\delta}_j^{(i)} \right) \equiv \left(x_{l,j}^{(k)}, k, l, \theta_{l,j}^{(k)}, \delta_{l,j}^{(k)} \right)$ is for variable

$x_{l,j}^{(k)}$ in $X^{(i)} =$

$$\left\{ x_{l,j}^{(k)} \mid \rho_{l,j}^{(k)} \equiv (\eta, \theta, \delta), \eta = i; j = 1, 2, \dots, \lambda_{l,k}, l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n \right\},$$

$\forall i \in V$, and $\gamma_i = |X^{(i)}|$.

Binary Variables

$$q_{j',j}^{(i)} \in \{0, 1\}, \forall j', j \in \{1, 2, \dots, \gamma_i\}, j \neq j'; \forall i \in V \quad (4.28)$$

$$\bar{q}^{(i)} \in \{0, 1\}, \forall i \in V \quad (4.29)$$

Constraints

$$y_{j'}^{(i)} + \tau_{\hat{\theta}_{j'}, \hat{\delta}_{j'}}^{(i)} + \tau_{\hat{\delta}_{j'}, \hat{\theta}_{j'}}^{(i)} \leq y_j^{(i)} + L \left(3 - q_{j',j}^{(i)} - z_l^{(k)} - z_{l'}^{(k')} \right), l = \hat{l}_j^{(i)}, \quad (4.30)$$

$$k = \hat{k}_j^{(i)}, l' = \hat{l}_{j'}, k' = \hat{k}_{j'}, \forall j', j \in \{1, 2, \dots, \gamma_i\}, j \neq j', \forall i \in V$$

$$\left(\sum_{j'=1, j' \neq j}^{\gamma_i} q_{j',j}^{(i)} \right) - 1 \leq L \left(1 - z_l^{(k)} \right), l = \hat{l}_j^{(i)}, k = \hat{k}_j^{(i)}, \quad (4.31)$$

$$j = 1, 2, \dots, \gamma_i, \forall i \in V$$

$$\left(\sum_{j'=1, j' \neq j}^{\gamma_i} q_{j,j'}^{(i)} \right) - 1 \leq L \left(1 - z_l^{(k)} \right), l = \hat{l}_j^{(i)}, k = \hat{k}_j^{(i)}, \quad (4.32)$$

$$j = 1, 2, \dots, \gamma_i, \forall i \in V$$

$$\sum_{j=1}^{\gamma_i} \sum_{\substack{j'=1, \\ j' \neq j}}^{\gamma_i} q_{j',j}^{(i)} = \sum_{j=1}^{\gamma_i} z_{l_j^{(i)}}^{\left(\hat{k}_j^{(i)}\right)} - \bar{q}^{(i)}, \forall i \in V \quad (4.33)$$

$$0 \leq L \left(1 - \bar{q}^{(i)}\right) + \sum_{j=1}^{\gamma_i} z_{l_j^{(i)}}^{\left(\hat{k}_j^{(i)}\right)} - 1, \forall i \in V \quad (4.34)$$

$$\tau_{1,\delta_j^{(i)}}^{(i)} - y_j^{(i)} \leq L \left(\sum_{\substack{j'=1, \\ j' \neq j}}^{\gamma_i} q_{j',j} + 1 - z_{l_j^{(i)}}^{\left(\hat{k}_j^{(i)}\right)} \right), j = 1, 2, \dots, \gamma_i, \forall i \in V \quad (4.35)$$

Constant

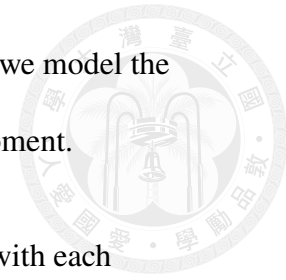
$$L : \text{a sufficiently large positive number} \quad (4.36)$$

Equation 4.30 states that when both the operations of j' -th tuple and the j -th tuple in $Y^{(i)}$ are present and the j -th is executed right after the j' -th, i.e., $q_{j',j}^{(i)} = 1$, the start time of the j -th operation $y_j^{(i)}$ should be later than the start time $y_{j'}^{(i)}$ plus the delivery time of the j' -th and the empty-car moving time from the j' -th to the j -th. Equations 4.31 and 4.32 impose the constraints on the operation of the j -th tuple such that there is at most one operation succeeding it and right after it.

Moreover, when there is more than one operation present in node i , i.e., $\bar{q}^{(i)} = 1$, the summation of $q_{j,j'}^{(i)}$ should be equal to the number of operations present in node i minus 1. Equation 4.34 tests if there is more than one operation present in node i . Finally, Eq. 4.35 demands the start time of the first operation executed in node i be later than the empty-car moving time from the initial P/D point to the pick-up P/D point of the operation.

4.2.4 Transfer Operation Constraints on Transfer Sites

This subsection explores the constraints on the transfer site buffer. First, we need to identify the pair-wise overlapping relationship to know whether a set of



operations overlaps. After pair-wise temporal overlaps are identified, we model the buffer size constraint on the count of concurrent operations at any moment.

Figure 4-8 demonstrates this idea. A set of operations overlap with each other in time if and only if they overlap pair-wisely in time. If we can define a binary variable $a_{i,j}^{(d)}$ in site e_d to represent that the i -th operation overlaps with the j -th, we can identify if a set of operations overlap with each other in time. On the left side, the three operations overlap with each other in time; therefore, all the pair-wise relationship $a_{i,j}^{(d)} = 1$. However, on the right side, the three operations do not overlap with each other temporally, which can be identified from the pair-wise relationships.

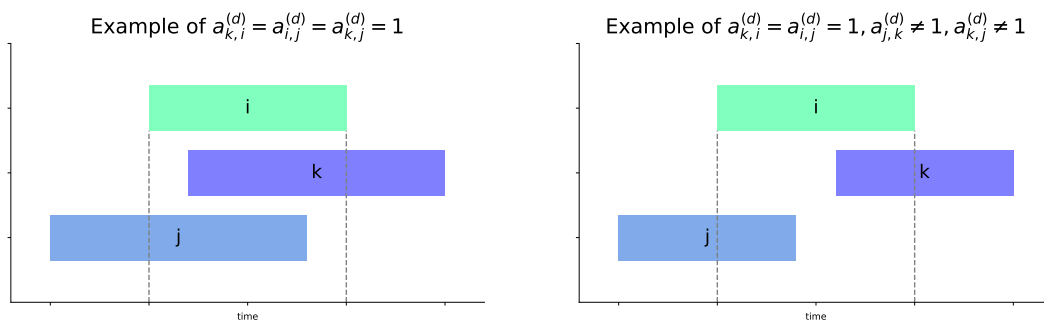


Figure 4-8. Illustration of two examples of three transfer operations in the site.

The constraints on a transfer site focus on the count of transfer operations concurrently under executions of transferring. Similar to defining the delivery operation set for a node, we will define the variable set representing the transfer operations to be executed for each transfer site. We have a set of operation start time

$$\tilde{X}^{(d)} = \left\{ \tilde{x}_{l,j}^{(k)} \mid e_{l,j}^{(k)} = e_d, j = 1, 2, \dots, \lambda_{k,l} - 1, l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n \right\}$$

for transfer site $e_d, d = 1, 2, \dots, r$.

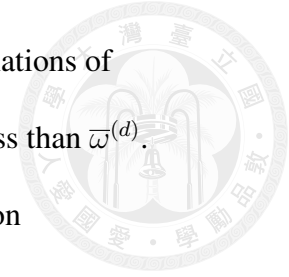
Similarly, we will reorganize the variables in $\tilde{X}^{(d)}$ for modeling convenience.

We collect the variables accompanied by their job indexes and path indexes as 4-tuples and index them from 1 to the number count of transfer operations on the transfer site e_d . Therefore, if the index for variable $\tilde{x}_{l,j'}^{(k)}$ mapped in the new set is j , the tuple is $(\tilde{y}_j^{(d)}, \tilde{k}_j^{(d)}, \tilde{l}_j^{(d)}, \tilde{s}_j^{(d)}) \equiv (\tilde{x}_{l,j'}^{(k)}, k, l, j')$. Let the set of the indexed tuples mapped from $\tilde{X}^{(d)}$ be $\tilde{Y}^{(d)} = \left\{ (\tilde{y}_j^{(d)}, \tilde{k}_j^{(d)}, \tilde{l}_j^{(d)}, \tilde{s}_j^{(d)}) \mid j = 1, 2, \dots, \tilde{\gamma}_d \right\}$, where $\tilde{\gamma}_d = |\tilde{X}^{(d)}|$ is the number of transfer operations conducted in transfer site e_d .

However, since the spanning time of a material in the site is indefinite, we need to collect another set to retrieve the end time of an operation in the transfer site. Let $\hat{Y}^{(d)} = \left\{ \hat{y}_j^{(d)} \equiv x_{\tilde{l}_j^{(d)}, \tilde{s}_j^{(d)}+1}^{(\tilde{k}_j^{(d)})} \mid j = 1, 2, \dots, \tilde{\gamma}_d \right\}$ be the end time of operations associated with $\tilde{Y}^{(d)}$.

In transfer site e_d , a transfer operation takes at least β_d time to reach the target node. The operation start times $\tilde{y}_j^{(d)}$ on the site are determined by the IP solver subject to the buffer size ω_d limit. In other words, at most ω_d interval variables overlap at any time. We introduce binary variables $a_{j',j}^{(d)} \in \{0, 1\}$ for indicating whether the start time of the j' -th transfer operation happens when the j -th is in the site. Therefore, $\tilde{y}_j^{(d)} < \tilde{y}_{j'}^{(d)} < \hat{y}_j^{(d)}$; $j, j' \in \{1, 2, \dots, \tilde{\gamma}_d\}, j \neq j'$ when $a_{j',j}^{(d)} = 1$. Also, we define the binary variable $\hat{a}_{j',j}^{(d)} \in \{0, 1\}$ to indicate if the start of the j' -th and the j -th operation are the same. Namely, $\tilde{y}_j^{(d)} = \tilde{y}_{j'}^{(d)}$ when $\hat{a}_{j',j}^{(d)} = 1$.

The buffer size constraints are implemented by limiting the number of overlapping operations to be less than $\omega_d + 1$ at any moment. Therefore, for any selection of $\omega_d + 1$ intervals, the summation of the pair-wise overlapping indicator $a_{j,j'}^{(d)}$ must be less than $\omega_d + (\omega_d - 1) + (\omega_d - 2) + \dots + 1 = \frac{\omega_d(\omega_d+1)}{2} = \bar{\omega}^{(d)}$.



Consequently, the buffer size constraints enumerate the combinations of $\omega_d + 1$ operations to limit the pair-wise overlap count to be strictly less than $\bar{\omega}^{(d)}$. Let $\Omega(m, \{1, 2, \dots, n\})$ be the operator that returns all of the selection combinations of m indexes from a set of n indexes as a list of sets of m indexes. The number of non-repeatable combinations of m indexes from n dissimilar indexes is $C_m^n = \frac{n!}{(n-m)!m!} = \binom{n}{m}$. For example, $\Omega(3, \{1, 2, 3, 4\})$ will yield $\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}$ as four combination sets. In our model, let $\tilde{b}_d = \binom{\tilde{\gamma}_d}{\omega_d + 1}$ be the number of combinations of selecting $\omega_d + 1$ operations. Let the index sets yielded from the operator $\Omega(\omega_d + 1, \{1, 2, \dots, \tilde{\gamma}_d\})$ be $A_1^{(d)}, A_2^{(d)}, \dots, A_{\tilde{b}_d}^{(d)}$. We need to establish constraints to avoid that for any index set $A_j^{(d)}, j = 1, 2, \dots, \tilde{b}_d$, all the operations in the set overlap with each other temporally, thus exceeding the site buffer limit.

Summary

Definition of Variable set

$$\tilde{Y}^{(d)} = \left\{ \left(y_j^{(d)}, \tilde{k}_j^{(d)}, \tilde{l}_j^{(d)}, \tilde{s}_j^{(d)} \right) \mid j = 1, 2, \dots, \tilde{\gamma}_d \right\}, \quad d = 1, 2, \dots, r \quad (4.37)$$

where the tuple $\left(\tilde{y}_j^{(d)}, \tilde{k}_j^{(d)}, \tilde{l}_j^{(d)}, \tilde{s}_j^{(d)} \right) \equiv \left(\tilde{x}_{l,j'}^{(k)}, k, l, j' \right)$ is defined for variable $\tilde{x}_{l,j'}^{(k)}$ in the set

$$\tilde{X}^{(d)} = \left\{ \tilde{x}_{l,j}^{(k)} \mid e_{l,j}^{(k)} = e_d, j = 1, 2, \dots, \lambda_{k,l} - 1, l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n \right\},$$

where $\tilde{\gamma}_d = |\tilde{X}^{(d)}|$.

$$\hat{Y}^{(d)} = \left\{ \hat{y}_j^{(d)} \equiv x_{\tilde{l}_j^{(d)}, \tilde{s}_j^{(d)} + 1}^{\left(\tilde{k}_j^{(d)} \right)} \mid j = 1, 2, \dots, \tilde{\gamma}_d \right\}, \quad d = 1, 2, \dots, r \quad (4.38)$$

Binary Variables

$$a_{j,j'}^{(d)} \in \{0, 1\}; \forall j, j' \in \{1, 2, \dots, \tilde{\gamma}_d\}, j \neq j', d = 1, 2, \dots, r \quad (4.39)$$



$$\hat{a}_{j',j}^{(d)} \in \{0, 1\}; \forall j, j' \in \{1, 2, \dots, \tilde{\gamma}_d\}, j \neq j', d = 1, 2, \dots, r \quad (4.40)$$

Constraints

$$\left\{ \begin{array}{l} \tilde{y}_j^{(d)} + \nu \leq \tilde{y}_{j'}^{(d)} + L \left(3 - a_{j,j'}^{(d)} - z_l^{(k)} - z_{l'}^{(k')} \right) \\ \tilde{y}_{j'}^{(d)} \leq \hat{y}_j^{(d)} - \nu + L \left(3 - a_{j,j'}^{(d)} - z_l^{(k)} - z_{l'}^{(k')} \right) \end{array} \right., \quad (4.41)$$

$$\left\{ \begin{array}{l} \tilde{y}_j^{(d)} \leq \tilde{y}_{j'}^{(d)} + L \left(3 - \hat{a}_{j,j'}^{(d)} - z_l^{(k)} - z_{l'}^{(k')} \right) \\ \tilde{y}_{j'}^{(d)} \leq \tilde{y}_j^{(d)} + L \left(3 - \hat{a}_{j,j'}^{(d)} - z_l^{(k)} - z_{l'}^{(k')} \right) \end{array} \right., \quad (4.42)$$

where $l = \tilde{l}_j^{(d)}, k = \tilde{k}_j^{(d)}, l' = \tilde{l}_{j'}^{(d)}, k' = \tilde{k}_{j'}^{(d)}, \forall j, j' \in \{1, 2, \dots, \tilde{\gamma}_d\}, j \neq j', d = 1, 2, \dots, r$.

$$\sum_{j \in A_i^{(d)}} \sum_{\substack{j' \in A_i^{(d)} \\ j' \neq j}} a_{j,j'}^{(d)} + \sum_{\substack{(j,j') \\ \in \Omega(2, A_i^d)}} \hat{a}_{j,j'}^{(d)} \leq \bar{\omega}^{(d)} - \nu + L \left(\omega_d + 1 - \sum_{j \in A_i^{(d)}} z_{\tilde{l}_j^{(d)}}^{(\tilde{k}_j^{(d)})} \right);$$

$$i = 1, 2, \dots, \tilde{b}_d, d = 1, 2, \dots, r \quad (4.43)$$

Constants

$$\bar{\omega}_d = \frac{\omega_d(\omega_d + 1)}{2} \quad (4.44)$$

$$\tilde{b}_d = \left(\begin{array}{c} \tilde{\gamma}_d \\ \omega_d + 1 \end{array} \right) \quad (4.45)$$

$$\nu : \text{a sufficiently small positive number} \quad (4.46)$$

$$L : \text{a sufficiently large positive number} \quad (4.47)$$

Equations 4.37 and 4.38 collect the start time and the end time of operations conducted in the transfer site d , respectively. When the j -th operation and the j' -th overlap, either $a_{j,j'}^{(d)}, a_{j',j}^{(d)}$, or $\hat{a}_{j,j'}^{(d)} (= \hat{a}_{j',j}^{(d)})$ would equal to 1. Equations 4.41 and

4.42 ensure the previous statement would hold true when both the j -th operation and the j' -th are present and overlap. Finally, to comply with the capacity limit, there must be strictly less than $\omega_d + 1$ operations overlapping with each other at any moment, which is realized by Eq. 4.43.

In the next section, we will introduce the metaheuristic algorithm for solving the MHNSP and particularly emphasize the solution encoding/decoding procedures and the details of the proposed Permutational Differential Evolution method.

4.3 Metaheuristic Algorithm for Material Handling Network Scheduling Problem

Metaheuristic algorithms are renowned for obtaining suitable solutions to extensive optimization problems in real-world applications. This section develops a metaheuristic algorithm for solving the MHNSP. First, a detailed illustration of the solution representation scheme, an integer array encoding scheme is presented. Subsequently, we outline the decoding procedures that lead to evaluating the makespan and violation metrics. Since material load buffering is involved in the problem, discrete-event simulation techniques are employed in the decoding algorithm.

Accompanied by the encoding scheme and decoding algorithm, a permutational differential evolution algorithm is lastly proposed for solving the MHNSP, which is regarded as a complex simulation-based optimization problem.



4.3.1 Solution Encoding and Decoding

The ingenuity of using metaheuristic algorithms to solve optimization problems lies in creating an efficient encoding scheme that minimizes computational resource use while searching for optimal solutions. Additionally, pairing the encoding scheme with an efficient decoding procedure is essential for achieving high-quality solutions with fewer computing resources.

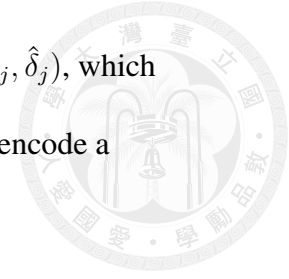
Different encoding schemes associated with dedicated decoding procedures can be developed to solve our MHNSPs. The following will illustrate a straightforward encoding scheme and a decoding procedure to determine the path selections and operation execution orders complying with precedence constraints for the MHNSP.

Encoding Scheme

The solution to the MHNSP is encoded as a permuted index array, commonly known as the permutation encoding. The indexes range from 1 to the total number of delivery operations decomposed from all candidate paths of all jobs. Let $\bar{\lambda}$ be the total number of delivery operations, $\bar{\lambda} = \sum_{k=1}^n \sum_{l=1}^{\sigma_k} \lambda_{l,k}$, where $\lambda_{l,k}$ is the number of operations on the l -th candidate path of job J_k . We aggregate all delivery operations to index them from 1 to $\bar{\lambda}$. Each operation is represented as a 6-tuple consisting of its properties. Let the indexed operation set be

$$W = \left\{ \hat{w}_j \equiv \left(\hat{k}_j, \hat{l}_j, \hat{s}_j, \hat{\eta}_j, \hat{\theta}_j, \hat{\delta}_j \right) \mid j = 1, 2, \dots, \bar{\lambda} \right\},$$

where \hat{k}_j is the job index of operation j , \hat{l}_j is the candidate path index, \hat{s}_j is the sequence index on the path, $\hat{\eta}_j$ is the execution node, $\hat{\theta}_j$ is the pick-up point index,



and $\hat{\delta}_j$ is the drop-off. The delivery operation j in W is $\rho_{\hat{l}_j, \hat{s}_j}^{(k_j)} \equiv (\hat{\eta}_j, \hat{\theta}_j, \hat{\delta}_j)$, which belongs to the path $\pi_{\hat{l}_j}^{(k_j)}$ in the candidate path set Π_{k_j} of job J_{k_j} . We encode a solution as a permuted index array

$$\mathbf{w} = [w_1 w_2 \cdots w_{\bar{\lambda}}], w_j \in \{1, 2, \cdots, \bar{\lambda}\}, w_j \neq w_{j'}, \forall j, j' \in \{1, 2, \cdots, \bar{\lambda}\}.$$

The permutation array, therefore, arranges the decoding sequence for these indexed delivery operations. That means operation w_j will be the j -th scheduled operation in the decoding procedure. This simple encoding scheme does not encode dedicated variables for path selections and transfer operations. Moreover, the permuted operations on the same path might violate precedence constraints, and no modeling for the buffer size constraints on transfer sites. These issues will be resolved in the decoding procedure, which is a constructive algorithm that generates a valid schedule for the problem.

Revisit the example introduced in the end of Section 3.1. There are two transportation jobs, where job J_1 has two candidate paths. Figure 4-9 shows the indexed operation set and a permuted index array for the example. According to the permuted index array, operations in the $\pi_2^{(1)}$ would be executed in the order $(2, 1, 4) \mapsto (1, 1, 2) \mapsto (4, 3, 1) \mapsto (3, 3, 2)$. However, this order sequence would violate the precedence constraints, that is, the operations in a path should be executed by the given sequence. In this case, it is $(1, 1, 2) \mapsto (2, 1, 4) \mapsto (3, 3, 2) \mapsto (4, 3, 1)$ to comply with actual material moving path.

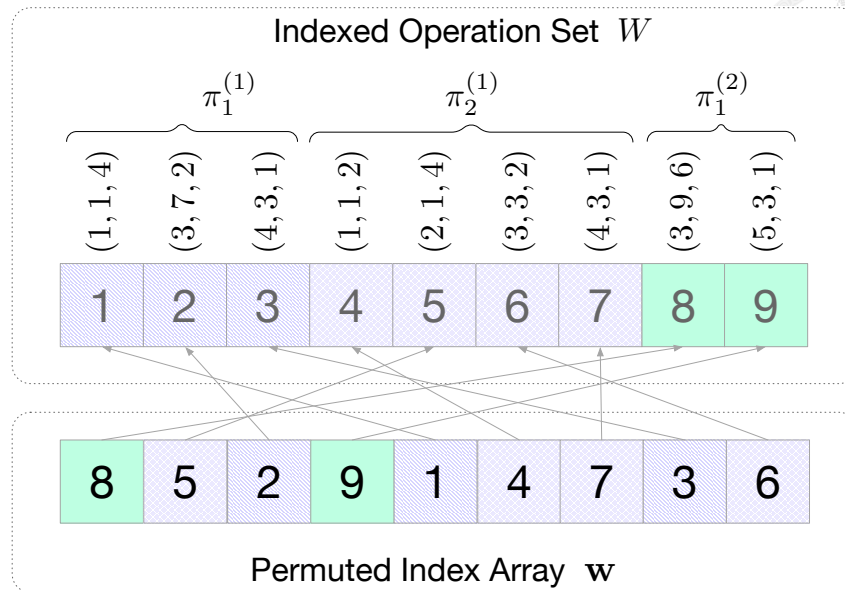
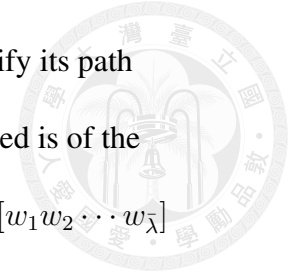


Figure 4-9. Illustration of two examples of three transfer operations in the site.

Decoding

The decoding procedure follows the generated sequence to deal with each delivery operation until the schedule is completed for objective value evaluation. We define l_k^* , $k = 1, 2, \dots, n$ as the selected path index for job J_k and initially set it to 0 for not being determined. During the decoding, the selection index is set to the path index of the first encountered operation of job J_k , i.e., $l_k^* \leftarrow \hat{l}_j$ if operation j in W is the first operation of job J_k dealt with. Once the selected path is set, successive operations on other paths of the same job are directly skipped without scheduling, since the path is not selected.

As mentioned before, the execution of delivery operations on a path must follow the precedence orders. The permuted operation sequence for a path might violate the precedence constraint. Therefore, we define a sequence index \hat{q}_k for each job J_k . The index starts from 1 and is forwarded once the preceding operation



is scheduled. Thus, in dealing with operation w_j of W , we only identify its path index and job index, i.e., \hat{l}_j and \hat{k}_j . The target operation to be scheduled is of the sequence index \hat{q}_k . In decoding the integer index array solution $\mathbf{w} = [w_1 w_2 \cdots w_\lambda]$ to the MHNSP, we will set start times of delivery operations on the selected path, i.e., $x_{l,j'}^{(k)}, j' = 1, 2, \cdots, \lambda_{k,l^*}$.

As indicated, the permuted index array represents a solution that defines the delivery operation execution orders that will not violate precedence constraints on operations of the same job and determine the candidate paths. In the first stage of the decoding procedure, we follow the given index array to select candidate paths and assign the execution sequences of the delivery operations executed on each node. In this stage, we decode the given integer array \mathbf{w} to construct the sequence of delivery operations selected for each node while discarding those unselected. Let the 3-tuple $\mathbf{b}_j^{(i)} \equiv (k_j^{(i)}, l_j^{(i)}, s_j^{(i)})$ be the j -th "execution" for node i , where $k_j^{(i)}$ is the job index of the j -th executed delivery operation, $l_j^{(i)}$ is the candidate path index of the operation, and $s_j^{(i)}$ is the operation sequence index on the path. Then, let $B^{(i)} = [\mathbf{b}_j^{(i)} \mid j = 1, 2, \cdots, \lambda^{(i)}]$ be the execution sequence of node i .

Therefore, the j -th executed delivery operation on node i is

$\rho_{l_j^{(i)}, s_j^{(i)}}^{(k_j^{(i)})} = (i, \theta_{l_j^{(i)}, s_j^{(i)}}^{(k_j^{(i)})}, \delta_{l_j^{(i)}, s_j^{(i)}}^{(k_j^{(i)})}) \equiv (i, \theta_j^{(i)}, \delta_j^{(i)})$, which belongs to the path $\pi_{l_j^{(i)}}^{(k_j^{(i)})}$ in the candidate path set Π_{k_j} of job J_{k_j} .

Figure 4-10 shows the conceptual idea of stage 1 in the decoding procedure. The path selection for each job is done by picking the first occurrence of the operation in the path for the job. In this example, the 5th operation shows up prior to the 2nd; therefore, the path $\pi_2^{(1)}$ is selected for job J_1 (refer to Fig. 4-9 for the

details of paths). The operation sequence in a path is amended simultaneously while selecting the path. Meanwhile, the sequence of delivery operations in nodes can also be identified. In this example, the 8th operation is ahead of the 6th in the encoding. As a result, the operations are executed in this order in node 3. In fact, when the decoding procedure is completed, this index array solution should yield the same result as solution D in Section 3.1.5. The Gantt chart was shown in Fig. 3-11 on page 38.

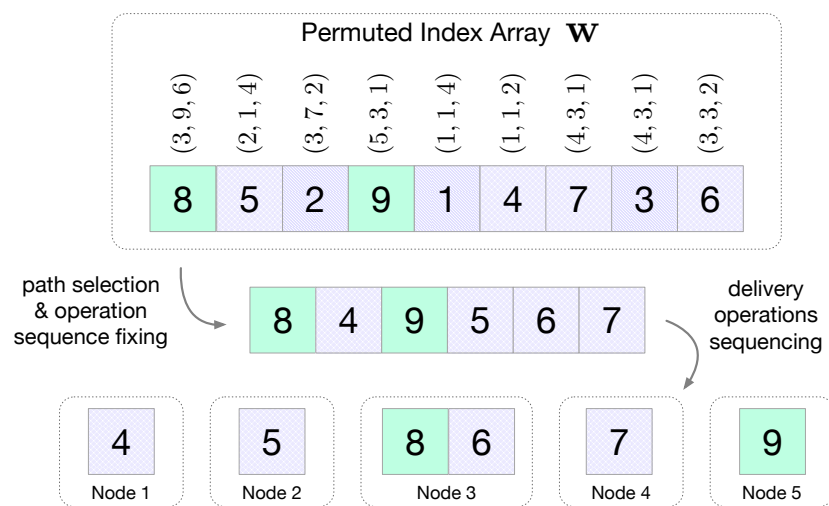
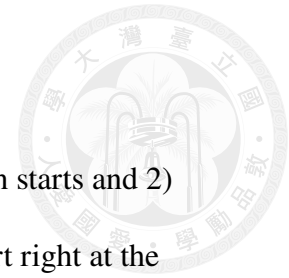


Figure 4-10. Illustration of the concept in the first stage of the decoding procedure.

Note that transfer operations associated with the previous delivery operation on a transfer site are *indefinite* since transfer operation times are not fixed. They are determined when the material loads are picked up in the following delivery operations. When no buffer vacancy is available on a transfer site, the succeeding delivery operation cannot start where the vehicle and the completed delivery operation job are blocked. As a result, it is *impossible to know the states of the material handling network beforehand*. Therefore, the decoding procedure employs a discrete-event simulation process with two discrete-time events to coordinate the sequence of delivery operations on nodes and maintain the buffer availabilities on



site.

Two indexed discrete events are defined: 1) a delivery operation starts and 2) a delivery operation ends. The succeeding transfer operation may start right at the end of the preceding delivery operation if there is a vacancy on the transfer site; otherwise, the current delivery operation is blocked until one of the delivery operations completes and starts its next delivery operation, thus yielding a vacancy. Note that all transfer operations end right at the start of the succeeding delivery operations that remove the load from the occupied buffer of the transfer site.

Notice that the discrete event is always associated with an “execution” on a node. Therefore, we define an event as a 4-tuple $(\hat{t}, \hat{y}, \hat{i}, \hat{j})$, where \hat{t} is the event time, $\hat{y} \in \{1, 2\}$ is the event type, and $\mathbf{b}_j^{(\hat{i})}$ is the execution associated. When $\hat{y} = 1$ the future event happens at time \hat{t} on node \hat{i} to start the \hat{j} -th delivery operation, $\hat{y} = 2$ to end the operation.

Discrete-event simulation techniques are executed in the second stage to construct the full schedule for all operations following the availability of operations and node resources. To bookkeep the operation execution sequence, let $z^{(i)}$ be the execution sequence index of the node i . Similarly, let z_k be the operation execution sequence index of job J_k . We bookkeep and update the available time $\nu^{(i)}$ of node i when it arrives at the drop-off point and the load starts its transfer operation. Conversely, we bookkeep and update the available times v_k of job J_k when the transfer operation had started for the minimal transfer time on the transfer site, where the succeeding operation can be processed.

Before the procedure starts, the future event list is constructed by inserting feasible operation-start events. Let the list be $F = [f_1 f_2 \cdots f_j f_{j+1} \cdots]$, a list of events arranged in ascending order of their event times, i.e., $\hat{t}_j \leq \hat{t}_{j+1}$. Then, the discrete event simulation starts to schedule delivery operations subject to precedence constraints on jobs, sequence constraints on nodes, and buffer size constraints on transfer sites as well. Each simulation step removes the head event f_1 from F for processing to update related system states. In event processing, new successive events might be generated and inserted back to F . The new events are inserted into the right places by checking their event times and maintaining the events in the correct order. The simulation procedure is, therefore, repeatedly removing the head event to process it and inserting back newly generated events until F is empty.

During the event processing, state variables, such as the job and node vehicle available times, are repeatedly updated while the start times of delivery operations, $x_{l_k^*, j'}^{(k)}$ and transfer operations, $\tilde{x}_{l_k^*, j'}^{(k)}$, are set step by step. However, when the deadlock happens, the event list F would become empty prematurely, where not all $x_{l_k^*, j'}^{(k)}$ and $\tilde{x}_{l_k^*, j'}^{(k)}$ are set.

A *deadlock* means that a resource in the system is required but it will never be released. Consider the network in Fig. 4-11. There are two nodes and a transfer site with a capacity limit of 1 in this network.

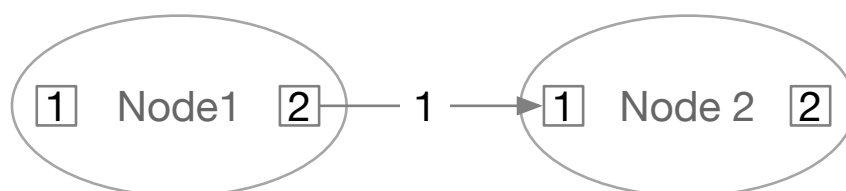
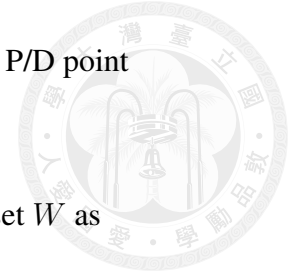


Figure 4-11. A material handling network for deadlock demonstration.



If we have two jobs, both starts from P/D point $p_1^{(1)}$ and ends at P/D point $p_2^{(2)}$. We have $\Pi_1 = \{\pi_1^{(1)} \equiv \langle (1, 1, 2), (2, 1, 2) \rangle\}$ and $\Pi_2 = \{\pi_1^{(2)} \equiv \langle (1, 1, 2), (2, 1, 2) \rangle\}$. We define the indexed operation set W as

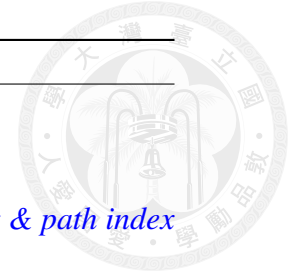
$$W = \left\{ \hat{w}_1 = (1, 1, 1, 1, 1, 2), \hat{w}_2 = (1, 1, 2, 2, 1, 2), \right. \\ \left. \hat{w}_3 = (2, 1, 1, 1, 1, 2), \hat{w}_4 = (2, 1, 2, 2, 1, 2) \right\}.$$

If the integer index array $\mathbf{w} = [1 \ 3 \ 4 \ 2]$, meaning that node 1 should execute the operation from the job J_1 first, while node 2 should execute the operation from the job J_2 first. However, this is impossible since the transfer site only has one buffer size. When the material load of job J_1 enters the site, the material load of job J_2 could not enter the site before the stored material of job J_1 is removed from the site. As a result, no material can enter node 2 eventually, thus causing a deadlock. When this happens, it is impossible to set the start time of every operation since some of them will never be executed. In this situation, the makespan cannot be calculated; hence, we introduce a violation metric to calculate the objective value by the number of operations not completed. The decoding procedures on a permuted index array are listed in Algorithm 8.

Algorithm 8 PermutationArrayDecoder (w)

-
- 1: $\mathbf{B} \leftarrow \{B^{(i)} \mid \forall i \in V\}; \mathbf{\Lambda} \leftarrow \{\lambda^{(i)} \mid \forall i \in V\};$
 - 2: \triangleright *Set of delivery operation start time*
 - 3: $\mathbf{X} \leftarrow \{x_{l,j}^{(k)} \mid j = 1, 2, \dots, \lambda_{k,l}, l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n\}$
 - 4: \triangleright *Set of transfer operation start time* \triangleleft
 - 5: $\tilde{\mathbf{X}} \leftarrow \{\tilde{x}_{l,j}^{(k)} \mid j = 1, 2, \dots, \lambda_{k,l} - 1, l = 1, 2, \dots, \sigma_k, k = 1, 2, \dots, n\}$
 - 6: \triangleright *Set of job completion time* \triangleleft
 - 7: ${}^c\mathbf{T} \leftarrow \{{}^c t_k \mid k = 1, 2, \dots, n\}$
 - 8: $B^{(i)} \leftarrow [], \lambda^{(i)} \leftarrow 0, \forall i \in V$
 - 9: $x' \leftarrow -1, \forall x' \in \mathbf{X}; \tilde{x}' \leftarrow -1, \forall \tilde{x}' \in \tilde{\mathbf{X}}; {}^c t' \leftarrow -1, \forall {}^c t' \in {}^c\mathbf{T};$
 - 10: \triangleright *Path selection & determine delivery operation sequence in nodes* \triangleleft
 - 11: SelectPathToSetOperationSequenceOnNodes($w, \mathbf{B}, \mathbf{\Lambda}$)
 - 12: \triangleright *Acquire operation start/end time & job completion time* \triangleleft
 - 13: SimulateOperationExecution($w, \mathbf{B}, \mathbf{\Lambda}, \mathbf{X}, \tilde{\mathbf{X}}, {}^c\mathbf{T}$)
 - 14: **return** CalculateObjectiveValue($\tilde{\mathbf{X}}, {}^c\mathbf{T}$) \triangleright *Calculate the objective value (makespan)*
-

Lines 8 and 9 initialize the execution sequence, number of operations in a node, delivery/transfer operation start time, and job completion time. We initialize the time with -1 to calculate the number of incomplete operations when the deadlock happens since a completed operation should have a start time greater than or equal to 0. The path for each job and the execution sequence in a node are then determined in Line 11. Afterward, Line 13 finds the start time of each delivery/transfer operation and the completion time for each job. Finally, the objective value based on the time acquired in the previous step is obtained in Line 14.



Algorithm 9 SelectPathToSetOperationSequenceOnNodes($\mathbf{w}, \mathbf{B}, \Lambda$)

```

1:  $\bar{\lambda} \leftarrow |\mathbf{w}|; q_k \leftarrow 1, \forall k = 1, 2, \dots, n;$ 
2: for  $j = 1, 2, \dots, \bar{\lambda}$  do
3:    $w \leftarrow w_j; k \leftarrow \hat{k}_w; l \leftarrow \hat{l}_w; \triangleright$  Get operation index, job index & path index
4:   if  $l_k^* = 0$  then
5:      $l_k^* \leftarrow l \triangleright$  Set selected path index
6:   else if  $l \neq l_k^*$  then
7:     goto 2 for next  $j$ 
8:    $j' \leftarrow q_k; q_k \leftarrow q_k + 1; \triangleright$  Update job sequence index
9:    $(i, \theta, \delta) \leftarrow \rho_{l, j'}^{(k)} \triangleright$  Identify delivery operation
10:   $B^{(i)} \leftarrow B^{(i)} \oplus (k, l, j'); \lambda^{(i)} \leftarrow \lambda^{(i)} + 1 \triangleright$  Append an operation to node i

```

Algorithm 9 is a procedure to find the selected path for each job and determine the execution sequence of operations in each node, i.e., $B^{(i)}, \forall i \in V$. Line 5 sets the path index to l_k^* if it is the first occurrence of the job J_k in \mathbf{w} . Lines 8 to 10 then append the operation if it belongs to the selected path of job J_k .

Algorithm 10 SimulateOperationExecution($w, B, \Lambda, X, \tilde{X}, {}^cT$)

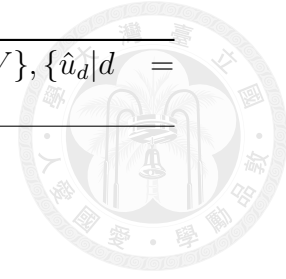
```

1:  $F \leftarrow []$   $\triangleright$  Event list
2:  $z^{(i)} \leftarrow 1, \nu^{(i)} \leftarrow 0; \forall i \in V$   $\triangleright$  Set node available time, & op. index
3:  $z_k \leftarrow 1; \nu_k \leftarrow g_k; k = 1, 2, \dots, n$   $\triangleright$  Set job available times & op. indexes
4:  $b_d \leftarrow 0; \hat{u}_d \leftarrow \omega_d; d = 1, 2, \dots, r$   $\triangleright$  Set initial blocked flags & vacancies for sites

5: foreach node  $i$  in  $V$  do
6:   if  $s_1^{(i)} = 1$  then  $\triangleright$  If it is also the 1st operation of a job
7:      $k \leftarrow k_1^{(i)}; \theta \leftarrow \theta_1^{(i)}; \tau \leftarrow \tau_{1,\theta}^{(i)}; \nu^{(i)} \leftarrow \tau;$ 
8:      $(\hat{t}, \hat{y}, \hat{i}, \hat{j}) \leftarrow (\max(\nu^{(i)}, \nu_k), 1, i, 1)$ 
9:      $F \leftarrow F \hat{+} (\hat{t}, \hat{y}, \hat{i}, \hat{j})$   $\triangleright$  Insert a start event
10:  repeat until  $F = []$ 
11:     $(t, y, i, j) \leftarrow f_1; F \leftarrow F \hat{-} f_1$   $\triangleright$  Get the head event
12:     $k \leftarrow k_j^{(i)}; l \leftarrow l_j^{(i)}; s \leftarrow s_j^{(i)}; \theta \leftarrow \delta_{l,s}^{(k)}; \delta \leftarrow \delta_{l,s}^{(k)};$ 
13:    if  $s \neq z_k$  then
14:      goto 10
15:    if  $y = 1$  then  $\triangleright$  operation start event
16:      StartAnOperation( $F, f_1, B^{(i)}, \nu_k, \nu^{(i)}, \{z_i | \forall i \in V\}, \{\hat{u}_d | d = 1, 2, \dots, r\}$ )
17:    else  $\triangleright$  ( $y = 2$ ) operation completion event
18:      CompleteAnOperation( $F, f_1, B^{(i)}, \nu_k, \nu^{(i)}, z_i, \{\hat{u}_d | d = 1, 2, \dots, r\}, \{b_d | d = 1, 2, \dots, r\}$ )
19:  until  $F = []$ 

```

In the discrete-event simulation, we try to determine the start time of every delivery/transfer operation and the completion time of each job. We insert the initial operation-start events if the operation is the first in both the job and the node. The procedure is described from lines 5 to 9. The simulation is conducted until the event list F becomes empty.



Algorithm 11 StartAnOperation($F, f_1, B^{(i)}, \nu_k, \nu^{(i)}, \{z_i | \forall i \in V\}, \{\hat{u}_d | d = 1, 2, \dots, r\}$)

```

1:  $(t, y, i, j) \leftarrow f_1;$ 
2:  $k \leftarrow k_j^{(i)}; l \leftarrow l_j^{(i)}; s \leftarrow s_j^{(i)}; \theta \leftarrow \delta_{l,s}^{(k)}; \delta \leftarrow \delta_{l,s}^{(k)};$ 
3: if  $\nu^{(i)} > t$  or  $\nu_k > t$  then
4:    $F \leftarrow F \hat{+} (\max(\nu^{(i)}, \nu_k), 1, i, j)$ 
5: else
6:    $x_{l,j'}^{(k)} \leftarrow t; z_i \leftarrow j;$  ▷ Set delivery operation start time & op. index
7:    $\nu^{(i)} \leftarrow t + \tau_{\theta,\delta}^{(i)}; \nu_k \leftarrow \nu^{(i)};$  ▷ Set node/job available time
8:    $F \leftarrow F \hat{+} (\nu^{(i)}, 2, i, j)$  ▷ Upload a completion event
9:    $z_k \leftarrow z_k + 1;$ 
10:  if  $s > 1$  then ▷ If the operation is not the 1st of the job
11:     $e_d \leftarrow \epsilon_{l,s-1}^{(k)}; \hat{u}_d \leftarrow \hat{u}_d + 1;$  ▷ Get the previous site & add a vacancy
12:    if  $b_d \neq 0$  then ▷ Release the blocked state
13:       $F \leftarrow F \hat{+} (t, 2, b_d, z_{\eta_{l,s-1}}^{(k)})$ 
14:       $b_d \leftarrow 0$ 

```

If an operation is ready to start, Line 7 sets the node available time $\nu^{(i)}$ to the time when the load arrives at its drop-off point, and a completion event is inserted with the event time of the node available time. If the operation is not the first of the job, we update the states of its previous transfer site. If the site was blocked, we insert a completion event to allow the blocked operation in the previous node to enter the site.

Algorithm 12 CompleteAnOperation($F, f_1, B^{(i)}, \nu_k, \nu^{(i)}, z_i, \{\hat{u}_d | d = 1, 2, \dots, r\}, \{b_d | d = 1, 2, \dots, r\}$)

```

1:  $(t, y, i, j) \leftarrow f_1$ ;
2:  $k \leftarrow k_j^{(i)}; l \leftarrow l_j^{(i)}; s \leftarrow s_j^{(i)}; c \leftarrow 0$ ;
3: if  $s = \lambda_{k,l}$  then
4:    ${}^c t_k \leftarrow t; c \leftarrow 1$ ;
5: else
6:    $e_d \leftarrow \epsilon_{l,s}^{(k)}; i' \leftarrow \eta_{l,s+1}^{(k)}$   $\triangleright$  Get the next site & node
7:   if  $b_d = 0$  then
8:     if  $\hat{u}_d > 0$  then  $\triangleright$  If the next site is available
9:        $\tilde{x}_{l,s}^{(k)} \leftarrow t; \hat{u}_d \leftarrow \hat{u}_d - 1; \nu_k \leftarrow t + \beta_d; c \leftarrow 1$ ;
10:       $F \leftarrow F \hat{+} \left( \max(\nu_k, \nu^{(i')}) , 1, i', z^{(i')} \right)$ 
11:     else
12:        $b_d \leftarrow i$ 
13: if  $c = 1$  and  $j < \lambda^{(i)}$  then
14:    $\nu^{(i)} \leftarrow t + \tau_{\delta_j^{(i)}, \theta_{j+1}^{(i)}}^{(i)}; z^{(i)} \leftarrow z^{(i)} + 1$ ;
15:    $F \leftarrow F \hat{+} \left( \max(\nu_k, \nu^{(i')}) , 1, i, z^{(i)} \right)$ 

```

If the operation is the last of the job, we set the job completion time.

Otherwise, we need to check the availability of the next site. When the next site is not blocked and available to enter (Line 8), we set the job available to the time when the minimum transfer time elapses and insert an operation-start event. Finally, if an operation is completed and it is not the last of the node, the next operation can try to enter the node.

Algorithm 13 CalculateObjectiveValue($\tilde{\mathbf{X}}, {}^c \mathbf{T}$)

```

1:  $\hat{\mathbf{X}} \leftarrow \tilde{\mathbf{X}} \cup {}^c \mathbf{T}; L \leftarrow 10^6$ ;
2: if  $y \neq -1, \forall y \in \hat{\mathbf{x}}$  then
3:    $C^{\max} \leftarrow \max({}^c \mathbf{T})$ 
4:   return  $C^{\max}$ 
5: else
6:   return  $\frac{|\{y | y = -1, \forall y \in \hat{\mathbf{x}}\}|}{|\hat{\mathbf{x}}|} \cdot L + L$ 

```



When every operation is completed—no deadlock occurs, the objective value is the makespan. If the deadlock happens, we calculate the penalty term by the fraction of incomplete operations times a large number L .

In the following subsection, we will introduce the Metaheuristic algorithm tailored for solving the Material Handling Network Scheduling Problem.

4.3.2 Permutational Differential Evolution Solver

Originally designed for continuous-valued optimization, Differential Evolution (DE) [22] has been found amenable to several real applications [1]. In this research, DE has been modified to adapt it for permutational tasks. Specifically, the crossover and mutation functions have been redefined to handle permutation encodings, making the DE suitable for solving the Material Handling Network Scheduling Problem. Moreover, our DE solver has also been modified for parallel computing. Due to its simple structure in an iteration, the parallelized version of DE will have less overhead than that of the genetic algorithm, which involves different stages in an iteration thus less efficient in parallel computing.

Crossover and Mutation

To apply Differential Evolution to permutation encodings, we incorporate the order-based crossover (OBX) and swap mutation techniques from the Genetic Algorithm.

The order-based crossover (OBX) process randomly selects C encoding positions of another solution w' to crossover with the base solution w . The order

of the values in these positions in w is replaced by their order of occurrence in the other solution w' . For example, in Fig. 4-12, the positions 3, 5, 6, and 7 are selected when $C = 4$. Therefore, the values 0, 3, 4, and 5 are to be crossed over. The remaining values, i.e., 1, 2, and 6, are kept in the order in the base solution. Finally, we fill in the selected values according to their order in the solution w' .

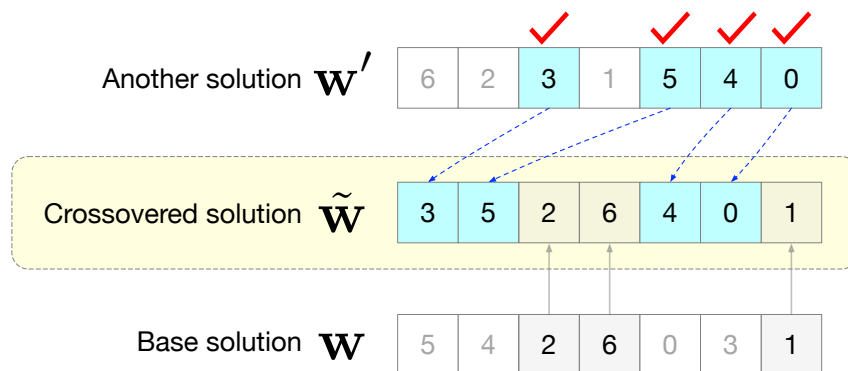


Figure 4-12. An example of order-base crossover.

In the swap mutation (Swap), two positions are randomly picked, and their values are swapped. This step is repeated several times to achieve the desired number of mutations M . The example in Fig. 4-13 selects the positions 3 and 6. Hence, the values in these positions, i.e., 2 and 4, are swapped.

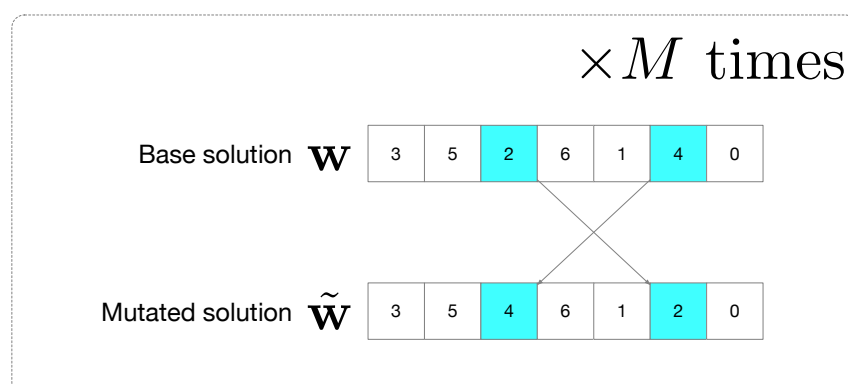
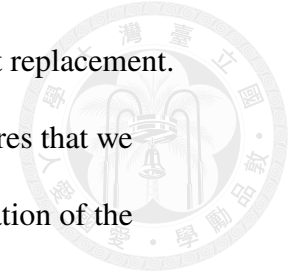


Figure 4-13. An example of swap mutation.

Algorithm 14 implements the OBX for parallel computing, where



$\text{RndChoice}(n, S)$ randomly selects n elements from the set S without replacement.

Since the solution \mathbf{w}' might change during the crossover, Line 7 ensures that we can traverse the solution \mathbf{w}' again when j exceeds $\bar{\lambda}$. The implementation of the swap mutation is listed in Algorithm 15.

Algorithm 14 $\text{ParallelOBX}(\mathbf{w} \equiv [w_1 w_2 \cdots w_{\bar{\lambda}}], \mathbf{w}' \equiv [w'_1 w'_2 \cdots w'_{\bar{\lambda}}], C)$

```

1:  $I \leftarrow \text{RndChoice}(C, \{1, 2, \dots, \bar{\lambda}\})$ 
2:  $\tilde{W} \leftarrow \{w'_i | \forall i \in I\}$   $\triangleright$  Values to be crossed over
3:  $\tilde{\mathbf{w}} \leftarrow \mathbf{w}; j \leftarrow 1$ 
4: for  $i = 1, 2, \dots, \bar{\lambda}$  do
5:   if  $\tilde{w}_i \in \tilde{W}$  then
6:     while  $w'_j \notin \tilde{W}$  do
7:        $j \leftarrow j + 1; j \leftarrow j \bmod \bar{\lambda}$ 
8:      $\tilde{w}_i \leftarrow w'_j$ 
9: return  $\tilde{\mathbf{w}}$ 

```

Algorithm 15 $\text{Swap}(\mathbf{w} \equiv [w_1 w_2 \cdots w_{\bar{\lambda}}], M)$

```

1:  $\tilde{\mathbf{w}} \leftarrow \mathbf{w}$ 
2: for  $i = 1, 2, \dots, M$  do
3:    $(j, j') \leftarrow \text{RndChoice}(2, \{1, 2, \dots, |\mathbf{w}|\})$ 
4:    $(\tilde{w}_j, \tilde{w}_{j'}) \leftarrow (\tilde{w}_{j'}, \tilde{w}_j)$ 
5: return  $\tilde{\mathbf{w}}$ 

```

Permutational Differential Evolution

This subsection details our approach to Permutational Differential Evolution.

Drawing inspiration from the original Differential Evolution, where crossover involves the difference between two parent solutions, we use the order-based crossover (OBX) and swap mutations (Swap) to combine the base solution with two other randomly selected parent solutions. If this new solution outperforms the base solution, it replaces it. The crossover rate \tilde{C} determines the portion of values

in the solution to be crossed over, while the mutation rate \tilde{M} decides the portion of values to be mutated. Let N be the population size, namely, the number of integer index arrays used for evolution. We also define the timeout \bar{T} for the DE to terminate the process after the processing time exceeds this timeout. The Permutational Differential Evolution process is outlined in Algorithm 16.

Algorithm 16 Permutational Differential Evolution

```

1:  $\mathbf{w}_i \leftarrow \text{Shuffle}([1\ 2 \cdots \bar{\lambda}]), i = 1, 2, \dots, N$ 
2:  $o_i \leftarrow \text{PermutationArrayDecoder}(\mathbf{w}_i), i = 1, 2, \dots, N$ 
3:  $t \leftarrow \text{time}()$ 
4: while  $\text{time}() - t \leq \bar{T}$  do
5:   parallel for  $i = 1, 2, \dots, N$ 
6:      $\mathbf{w}' \leftarrow \mathbf{w}_i$   $\triangleright$  Temporary solution
7:      $(j, j') \leftarrow \text{RndChoice}(2, \{1, 2, \dots, N\} \setminus \{i\})$ 
8:      $\mathbf{w}' \leftarrow \text{ParallelOBX}(\mathbf{w}', \mathbf{w}_j, \lceil \frac{\tilde{C} \cdot \bar{\lambda}}{2} \rceil)$ 
9:      $\mathbf{w}' \leftarrow \text{ParallelOBX}(\mathbf{w}', \mathbf{w}_{j'}, \lceil \frac{\tilde{C} \cdot \bar{\lambda}}{2} \rceil)$ 
10:     $\mathbf{w}' \leftarrow \text{Swap}(\mathbf{w}', \lceil \tilde{M} \cdot \bar{\lambda} \rceil)$ 
11:     $\tilde{o} \leftarrow \text{PermutationArrayDecoder}(\mathbf{w}')$ 
12:    if  $\tilde{o} < o_i$  then  $\triangleright$  Replace the base solution if the temporary is better
13:       $o_i \leftarrow \tilde{o}; \mathbf{w}_i \leftarrow \mathbf{w}'$ 
14:  $i^* \leftarrow \arg \min_{i=1,2,\dots,N} \{o_i\}$ 
15: return  $o_{i^*}, \mathbf{w}_{i^*}$   $\triangleright$  Return the best objective value and solution

```

Line 1 initializes the integer index array solutions, where $\text{Shuffle}(X)$ randomly permutes the elements in the list X . Line 3 saves the current time. In Line 5, **parallel for** means that the process in the loop, that is, the evolution of an index array solution, is executed in parallel on different CPU cores. Line 7 randomly selects two other solutions \mathbf{w}_j and $\mathbf{w}_{j'}$ to be crossed over with the temporary solution \mathbf{w}' . Finally, Line 12 replaces the base solution \mathbf{w} if the temporary solution \mathbf{w}' is better.

4.4 Summary

This section has presented an in-depth description of the three proposed methods to address the MHNSP: Constraint Programming, Integer Programming, and Metaheuristic approaches.

For Constraint Programming and Integer Programming, we initially focused on elucidating the methods to ascertain the makespan. We then delved into the intricacies of formulating constraints relevant to transportation jobs, nodes, and transfer sites. Regarding the Metaheuristic approach, we explored the encoding and decoding procedures to efficiently evaluate solutions and proposed Permutational Differential Evolution to find the optimal solution for the MHNSP.

The forthcoming chapter is dedicated to conducting extensive numerical analyses to assess the effectiveness and robustness of the three proposed methodologies.



Chapter 5



Numerical Tests and Result Discussion

Numerical test problems are required to evaluate the performance of different solving methods and the solutions obtained. This chapter provides a comprehensive examination of these aspects. First, we introduce the implemented solving system for the MHNSP. Next, we conduct **four numerical tests** and show their results. These tests include a *DE parameter-tuning experiment via Taguchi method*, a *model performance comparison test*, an *extra-large problem test*, and an *identical request test*. The source code of our solver implementations is published on GitHub at <https://github.com/markmarkchen/Material-Handling-Network-Scheduling-Problem-Solvers>.

5.1 Solving Method Implementations

We introduce the implemented solvers for the MHNSP in the section. We use the Python API of IBM[®] ILOG[®] CPLEX[®] v22.1.1 to construct the CP and IP models for solving the problems. We develop the DE solver as a Python package that uses NumPy and Numba packages and enables parallel computing via a JIT compiler to convert the Python code into machine codes. Therefore, our parallel-computing DE solver is being comparable with the highly-optimized ILOG[®] solvers. Finally, all of our tests are run on a computer with AMD Ryzen 5 3600 CPU and 64G DDR4-3600 RAM.

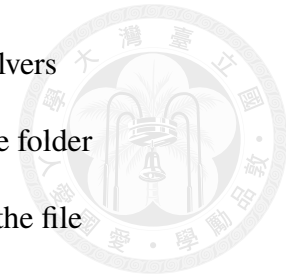


Figure 5-1 shows the manifest directory of our implemented solvers published on GitHub. We put all the implemented solver scripts in the folder MaterialHandlingNetworkSchedulingProblemSolvers. In this folder, the file BenchmarkParser.py provides a class to parse the problem files shown in Section 3.2.4 and returns a benchmark problem object. When solving the problem using either solver, the problem object is passed to each solver.

Outside the solver folder, we have the script Benchmark Generator (json).py that generates the numerical test problems specified by the user, where parameters are listed in the file named benchmark config.yml discussed in Section 3.2.4. Finally, all the tests are done by the respective scripts named with Test Platform. In these scripts, the corresponding problem files are parsed and passed to the solvers. These scripts establish a platform that can automatically conduct the tests and record the results. The screenshot of the published source code on GitHub is shown in Fig. 5-2. Readers are welcome to run the codes to reproduce the results, with IBM® ILOG® installed.

```
 /
├── MaterialHandlingNetworkSchedulingProblemSolvers
│   ├── MetaheuristicAlgorithmLibrary
│   ├── __init__.py
│   ├── BenchmarkParser.py
│   ├── ConstraintProgrammingSolver.py
│   ├── IntegerProgrammingSolver.py
│   └── PermutationalDifferentialEvolutionSolver.py
├── benchmark config.yml ..... USER-DEFINED PARAMETERS
├── Benchmark Generator (json).py
├── Test Platform (Taguchi).py ..... TEST #1
├── Test Platform (General).py ..... TEST #2
├── Test Platform (Extra).py ..... TEST #3
└── Test Platform (Identical).py ..... TEST #4
```

Figure 5-1. Directory structure of the implemented solvers.

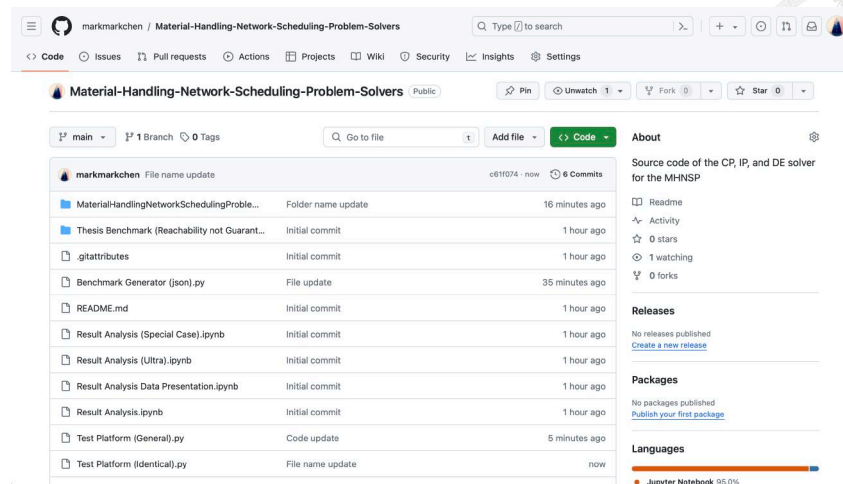


Figure 5-2. Screenshot of the source code of solver implementations on GitHub.

5.2 Numerical Tests and Result Discussion

We conduct *four numerical tests* to study the MHNSP and the proposed solving methods. First, we conduct a test for DE parameter-tuning via the Taguchi method. This test identifies the best parameters for each problem scale and problem type. Second, we run a model performance comparison test. The proposed CP, IP, and the tuned DE model are compared in this test for solving 360 randomly generated problems in different scales and types. The purposes of this test are to know the strengths and weaknesses of each solver in solving different problem scales and types and to compare the makespan reductions when the problem type changes from fixed to flexible on the same network, i.e., having routing flexibility. The third test compares the performance of the CP model and the DE model in extra-large problems. This test evaluates their performance in real applications. Finally, we will conduct a special test where all the requests share the same start/end points. This test is to verify that path selection options are indeed important and can drastically reduce the makespan.



5.2.1 Differential Evolution Model Parameter-tuning Experiment via the Taguchi method

The Taguchi method [23] is widely used in the design of experiments to reduce the number of experiments compared to the full factorial design. In this test, we try to identify the best parameters for the DE model. We test the numerical test problems in $1/\infty$, $1/n$, n/∞ , and n/n types, and there are three scales for each type: small, medium, and large. Therefore, a total number of 12 different problem settings are tested to find the best DE parameters for each of the settings. Note that the problem attribute and notations were introduced in Section 3.2.4 and summarized in Tables 3-2 and 3-3 on pages 50 and 54, respectively.

Three primary parameters for the DE model are population size N , mutation rate \tilde{M} , and crossover rate \tilde{C} . We choose three value levels for each parameter as listed in Table 5-1. For solving the fixed problems, we *empirically* set the timeout for DE computing to 2 seconds. For flexible problems, the timeout is 20 seconds.

Table 5-1. Design level for each DE parameter.

Parameter	Level		
	1	2	3
Population Size N	8	64	128
Mutation Rate \tilde{M}	0.001	0.01	0.1
Crossover Rate \tilde{C}	0.050	0.10	0.2

To select the best level for each parameter in a problem setting, we design the experiments using the L9 orthogonal array of the Taguchi method. There are 9 experiments, and we conduct 5 independent runs for each experiment. Table 5-2 shows the details for each experiment and the corresponding parameter level

settings. In the table, each $y_{i,j}$ represents the best makespan obtained by our DE solver.

Table 5-2. Taguchi L9 design.

Exp.	Parameter level			Parameter level value			Run					Performance
	N	\tilde{M}	\tilde{C}	N	\tilde{M}	\tilde{C}	Run 1	Run 2	Run 3	Run 4	Run 5	SNR
1	1	1	1	8	0.001	0.05	$y_{1,1}$	$y_{1,2}$	$y_{1,3}$	$y_{1,4}$	$y_{1,5}$	SNR_1
2	1	2	2	8	0.010	0.10	$y_{2,1}$	$y_{2,2}$	$y_{2,3}$	$y_{2,4}$	$y_{2,5}$	SNR_2
3	1	3	3	8	0.100	0.20	$y_{3,1}$	$y_{3,2}$	$y_{3,3}$	$y_{3,4}$	$y_{3,5}$	SNR_3
4	2	1	2	64	0.001	0.10	$y_{4,1}$	$y_{4,2}$	$y_{4,3}$	$y_{4,4}$	$y_{4,5}$	SNR_4
5	2	2	3	64	0.010	0.20	$y_{5,1}$	$y_{5,2}$	$y_{5,3}$	$y_{5,4}$	$y_{5,5}$	SNR_5
6	2	3	1	64	0.100	0.05	$y_{6,1}$	$y_{6,2}$	$y_{6,3}$	$y_{6,4}$	$y_{6,5}$	SNR_6
7	3	1	3	128	0.001	0.20	$y_{7,1}$	$y_{7,2}$	$y_{7,3}$	$y_{7,4}$	$y_{7,5}$	SNR_7
8	3	2	1	128	0.010	0.05	$y_{8,1}$	$y_{8,2}$	$y_{8,3}$	$y_{8,4}$	$y_{8,5}$	SNR_8
9	3	3	2	128	0.100	0.10	$y_{9,1}$	$y_{9,2}$	$y_{9,3}$	$y_{9,4}$	$y_{9,5}$	SNR_9

The idea of the Taguchi method is to calculate the signal-to-noise ratio (SNR) for each experiment, which is a performance indicator. The SNR for each experiment is

$$SNR_i = -10 \cdot \log \left(\frac{1}{N_i} \sum_{j=1}^{N_i} y_{i,j}^2 \right), \quad i = 1, 2, \dots, 9. \quad (5.1)$$

After obtaining the SNR for each experiment, we can estimate the effect of each level of the parameter by averaging out the SNRs of experiments using that level of design. For example, the effects of the three value levels of the mutation rate can be calculated by

$$T_{2,1} = \frac{(SNR_1 + SNR_4 + SNR_7)}{3}, \quad (5.2)$$

$$T_{2,2} = \frac{(SNR_2 + SNR_5 + SNR_8)}{3}, \quad (5.3)$$

$$T_{2,3} = \frac{(SNR_3 + SNR_6 + SNR_9)}{3}. \quad (5.4)$$

We can calculate the effects of the other two parameters, N and \tilde{C} , using the corresponding SNR terms. Finally, when the 9 effects of different levels of



parameters are obtained, the best value level for each parameter can be identified as shown in Table 5-3.

Table 5-3. Taguchi L9 design parameter selection.

Parameter	Level			Selected Level
	1	2	3	
N	$T_{1,1}$	$T_{1,2}$	$T_{1,3}$	$\arg \max_{j=1,2,3} (T_{1,j})$
\tilde{M}	$T_{2,1}$	$T_{2,2}$	$T_{2,3}$	$\arg \max_{j=1,2,3} (T_{2,j})$
\tilde{C}	$T_{3,1}$	$T_{3,2}$	$T_{3,3}$	$\arg \max_{j=1,2,3} (T_{3,j})$

The raw results are shown in Table B-1 in Appendix B. We can identify the best parameters for each problem setting by repeating the aforementioned procedure 12 times. The result is shown in Table 5-4. For small and medium-scale problems, there is no difference between each parameter setting. However, in large-scale problems, the best crossover rate and the mutation rate are higher than the minimum value levels.

Table 5-4. Best parameters of each problem setting.

Scale	Type	Population Size N	Mutation Rate \tilde{M}	Crossover Rate \tilde{C}
Small	$1/\infty$	8	0.001	0.050
	$1/n$	8	0.001	0.050
	n/∞	8	0.001	0.050
	n/n	8	0.001	0.050
Medium	$1/\infty$	8	0.001	0.050
	$1/n$	8	0.001	0.050
	n/∞	8	0.001	0.050
	n/n	8	0.001	0.050
Large	$1/\infty$	64	0.010	0.100
	$1/n$	64	0.010	0.100
	n/∞	8	0.010	0.100
	n/n	8	0.100	0.100

5.2.2 Model Performance Comparison Test

This test evaluates the performance of the three proposed models in solving problems of different scales and types. Moreover, we want to know the average makespan reduction when switching from a fixed problem to a flexible one in the same network, i.e., considering more candidate paths for each job.

We generated 30 networks on small, medium, and large scales, for a total of 90 networks. For each network, we consider four types of problems: $1/\infty$, $1/n$, n/∞ , and n/n types, as described in the previous test. Therefore, a total number of **360 randomly generated numerical test problems** are generated for this test.

The parameters found in the previous test are used to execute the DE model in the corresponding problem setting, and five runs are conducted. We set the DE timeout to 2 seconds and 20 seconds for fixed and flexible problems, respectively, following the previous test. For the CP and IP models, the timeout is set to 60 seconds for all types and scales of numerical test problems.

Tables B-2 and B-3 in Appendix B list the obtained makespan and solving time used in solving these problems. Consolidating the obtained data of the 30 networks of each problem scale/type, the hit rates and average solving time are listed in Table 5-5. The hit rates for the CP and IP models are the count ratio of obtaining the global minimum among the 30 networks. For the DE model, the hit rate is the ratio of runs out of the five runs that match or outperform the better makespan obtained by the CP and IP models. Finally, the average solving time is calculated by the mean of the solving time for the 30 networks in each scale/type.

Table 5-5. Hit rate and average solving time of model performance comparison test.

Scale	Type	Hit Rate			Average Solving Time (s)		
		CP	IP	DE	CP	IP	DE
Small	1/∞	1.000	0.967	1.000	0.373	0.020	2.000
	n/∞	1.000	1.000	1.000	0.811	0.518	20.001
	1/n	1.000	0.967	1.000	0.469	0.033	2.000
	n/n	1.000	1.000	1.000	0.928	0.849	20.001
Medium	1/∞	1.000	1.000	0.967	1.515	0.243	2.001
	n/∞	0.767	0.433	0.767	26.509	40.723	20.000
	1/n	1.000	1.000	1.000	1.656	0.778	2.001
	n/n	0.767	0.233	0.800	25.600	49.820	20.000
Large	1/∞	1.000	0.967	1.000	1.202	2.068	2.002
	n/∞	0.467	0.100	0.767	43.501	55.568	20.000
	1/n	1.000	1.000	1.000	1.341	0.112	2.001
	n/n	0.500	0.100	0.833	42.912	56.701	20.001

The numerical results shown indicate the performances of the three solvers in different problem types. First, in the fixed problems, the mathematical programming solvers, CP and IP, almost always find the optimal solutions regardless of the problem scale. Moreover, the IP solver is much faster than the CP solver in finding the optimal solution for fixed problems. In some cases, the difference is several orders of magnitude in speed.

Second, the mathematical programming solvers still perform well in small-sized flexible problems. However, the performance drops significantly in medium and large-scale problems. The performance degradation of the IP solver is notable, as it only finds the optimal solutions in one-tenth of the large n/n

problems. In contrast, the CP solver does not show this performance loss. Overall, the CP solver performs better than the IP solver in flexible problems than in solution qualities and optimization speed. Surprisingly, in the most difficult problems—large n/n , the CP solver still finds the optimal solutions for half of them. This indicates its potential for real applications. In the next test, we will compare the performance of the CP and DE models in solving extra-large problems.

Finally, the Permutational Differential Evolution method performs well across all types/settings of problems. It excels in solving medium- and large-scale flexible problems, outperforming the mathematical programming solvers in terms of optimality hit rate and computing speed.

Makespan Reduction from Fixed to Flexible Problems on the Same Network

Since the fixed problems are special cases derived from the flexible counterparts, having path flexibility (more candidate paths) for each job is expected to yield a makespan reduction. Let ${}^1C^{\max}$ and ${}^n C^{\max}$ be the makespans obtained from the fixed and flexible problems of the same network, respectively. We define the makespan reduction as

$$\frac{{}^1C^{\max} - {}^n C^{\max}}{{}^1C^{\max}}.$$

We calculate the average and the standard deviation of the 30 makespan reduction values from the 30 problems (networks) of each problem scale and site capacity settings. The data are listed in Table 5-6.

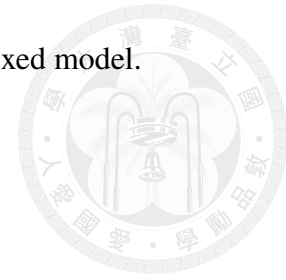


Table 5-6. Makespan improvement of the flexible model over the fixed model.

Capacity Type	Scale	Average C^{\max} Reduction	STD
∞	Small	0.029	0.060
	Medium	0.089	0.089
	Large	0.138	0.098
n	Small	0.029	0.061
	Medium	0.090	0.087
	Large	0.132	0.100

The result indicates a limited average reduction can be achieved in small-scale problems. However, as the problem scale increases, the average reduction also increases, exceeding 13% in large-scale problems. This is significant because the jobs and networks are randomly generated in our numerical test problems; therefore, some of the network layouts are unreasonable. In real-world scenarios, the average reduction should be much higher than our results.

5.2.3 Extra-large Problem Test

As shown in the previous test, the CP model performed well in solving the large n/n problems. This test intends to find its capability for extra-large problems.

We generate 5 extra-large problems by the user-defined parameters listed in Table 3-2. The timeout for the CP model is one hour. Similarly, we also conduct 5 independent runs of the DE model on these problems. The parameter settings follow the values obtained in the Taguchi experiment on the large n/n type problem in Table 5-4. The timeout for the DE solver is 12 minutes. The makespans obtained in the test are shown in Table 5-7. The gap is the difference between the

best makespan found in the DE model and the makespan from the CP solver divided by the CP makespan.

Table 5-7. Makespan comparison between CP and DE solvers in Extra-large Problems.

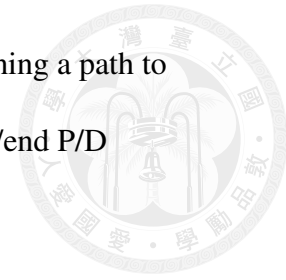
Net. ID	CP	DE-1	DE-2	DE-3	DE-4	DE-5	Gap
U-1	3978	4057	4045	4057	4045	4090	0.017
U-2	2683	4075	3737	4045	3891	3959	0.393
U-3	2422	3324	3329	3568	3439	3491	0.372
U-4	3047	3928	3969	4095	4043	3939	0.289
U-5	3086	3830	3802	3678	3829	3679	0.192

Surprisingly, the CP model outperforms the DE model in every problem.

This finding showcases that Constraint Programming has a high potential for solving problems in real applications. Moreover, when the solving time is limited, the CP model should be the preferred one to use. However, this also casts doubt on the previous test about whether the 1-minute timeout for the CP model is enough to obtain satisfying results in large problems. We will leave this to the future work.

5.2.4 Identical Request Test

In our last test, we set up a special case where the transportation requests are identical. We generated a large-scale network with finite buffer sizes, shown in Fig. 3-12 (c) on page 52. Three types of problems, $1/n$, $1'/n$, and n/n , are generated for the test. Notice that the first two are fixed problems while the last is a flexible one. Since the $1'/n$ problem randomly assigns a candidate path to each job, we generate 10 different instances for the test, each with different path assignments. The purposes of this test is to check whether multiple candidate paths for each job



are necessary or if we can emulate a similar result by randomly assigning a path to each job. In this test, five identical requests are specified, whose start/end P/D points of the job are $p_{17}^{(5)}$ and $p_{33}^{(2)}$.

Following similar settings in the previous tests, our DE model conducts 5 independent runs in each problem and uses the tuned parameters for the large n/n type problems. The timeout for the DE model is also set to 2 and 20 seconds for fixed and flexible problems. Similarly, the timeout for the CP and IP models is 60 seconds. Note that the network, requests, and site buffer settings are all the same in this test. The result is shown in Table 5-8.

Table 5-8. Numerical results for the identical request test.

Type	Objective Value (makespan)							Solving Time (s)						
	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5
$1/n$	398	398	398	398	398	398	398	3.80	0.31	2.00	2.00	2.00	2.00	2.00
$1'/n$ (1)	377	377	377	377	377	377	377	7.52	1.25	2.00	2.00	2.00	2.00	2.00
$1'/n$ (2)	328	328	328	328	328	328	328	4.24	0.20	2.00	2.00	2.00	2.00	2.00
$1'/n$ (3)	340	340	340	340	340	340	340	6.06	0.55	2.00	2.00	2.00	2.00	2.00
$1'/n$ (4)	292	292	292	292	292	292	292	4.12	0.31	2.00	2.00	2.00	2.00	2.00
$1'/n$ (5)	288	288	288	288	288	288	288	2.45	0.24	2.00	2.00	2.00	2.00	2.00
$1'/n$ (6)	316	316	316	316	316	316	316	3.90	0.77	2.00	2.00	2.00	2.00	2.00
$1'/n$ (7)	284	284	284	284	284	284	284	3.31	0.54	2.00	2.00	2.00	2.00	2.00
$1'/n$ (8)	379	379	379	379	379	379	379	3.87	0.31	2.00	2.00	2.00	2.00	2.00
$1'/n$ (9)	344	344	344	344	344	344	344	6.34	0.60	2.00	2.00	2.00	2.00	2.00
$1'/n$ (10)	325	325	325	325	325	325	325	3.82	0.57	2.00	2.00	2.00	2.00	2.00
n/n	256	-	264	261	270	274	256	60.07	-	20.00	20.00	20.00	20.00	20.00

As expected, the fixed problem cases are trivial for the methods to obtain the global optimum. However, in the flexible problem, the IP model could not even find a feasible solution before the timeout. This is surprising because there are only 5 jobs in the problem, compared to 12 jobs for the problems in the model

comparison test. Table 5-9 shows the reorganized results for this test.

Table 5-9. Reorganized numerical results for the identical request test.

Type Solver	1/n			1'/n (10 instances)			n/n		
	CP	IP	DE (5)	CP	IP	DE (5)	CP	IP	DE (5)
Average C^{\max}	-	-	398.00	327.30	327.30	327.30	-	-	265.00
Minimum C^{\max}	398	398	398	284	284	284	256	-	256
(Avg.) Solving Time (s)	3.80	0.31	2.00	4.56	0.53	2.00	60.07	-	20.00

The result indicates that path selection matters in completing the jobs earlier.

Even when the path is randomly assigned, the makespan decreases significantly,

from 398 to 327.3 on average. Ultimately, when the candidate paths are all

available in the n/n problem type, the best makespan is 256.

The Gantt charts of the operations of nodes and jobs, respectively, for the

fixed problem and the flexible one are compared in Figs. 5-3 and 5-4.

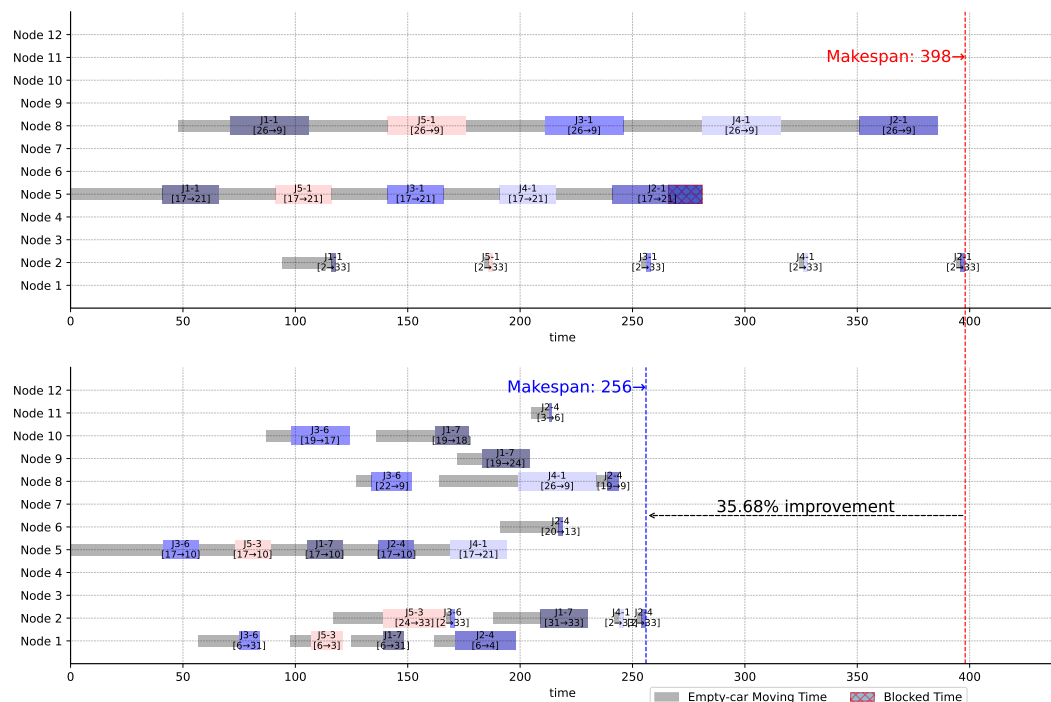


Figure 5-3. Node Gantt charts yielded from the fixed and flexible problems.

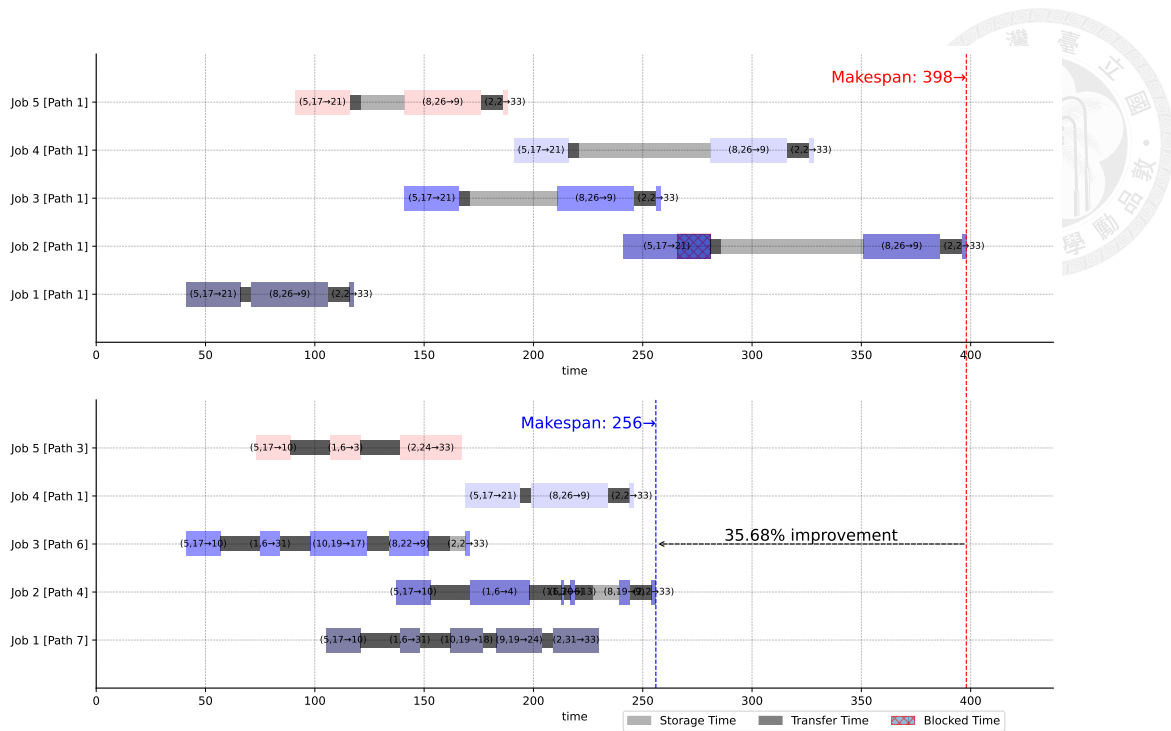


Figure 5-4. Job Gantt charts yielded from the fixed and flexible problems.

From the node utilization perspective shown in Fig. 5-3, the flexible solution yields a more balanced workload distribution among nodes. Figure 5-4 shows the Gantt blocks of delivery and transfer operation executed in turns for each job. Note that the transfer operations are much shorter in the flexible problem. Almost every transfer operation ends with the duration of the transfer time requirement. In contrast, the solution for the fixed problem has lengthy waiting periods in transfer sites. Moreover, it is interesting that only one of the jobs (job 4) prefers the fastest path (path 1) in the flexible problem. This example test verifies that the material handling system is not efficient when every transportation job selects the shortest/fastest path. This worst strategy has been widely adopted in the conventional real-time job dispatching systems. From the above test, we confirm that path selection optimization models should be adopted in a smart material handling system to complete the jobs with the minimal makespan.

5.3 Summary

This chapter introduced the implementation of our models and showed the numerical results from four different tests. In the DE parameter-tuning test, we tried to find the best DE parameter settings for different problem types using the Taguchi method. In the model performance comparison test, we generated 360 numerical test problems to test the proposed CP, IP, and DE models. The results showed that IP performed best in fixed problems, while CP stood out in small and medium flexible problems. The powerfulness of DE was only revealed in the medium and large n/n type of problems. In the extra-large problem test, we tested the performance of the CP and the DE models in extremely complex networks with numerous jobs. The results showed that when given enough time, the CP model could find much better solutions than the DE model. Finally, we tested the identical request problem to show that path selection is crucial in decreasing the makespan.





Chapter 6

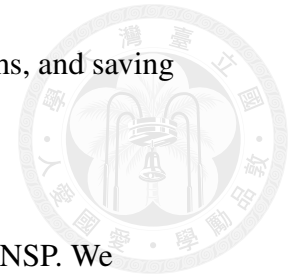
Conclusion and Future Work Suggestion

In this research, we have rigorously defined the Material Handling Network Scheduling Problem (MHNSP), where the goal is to minimize the makespan for a given set of transportation jobs. We have proposed a Constraint Programming model, an Integer Programming model, and a Permutational Differential Evolution model to solve the MHNSP. To evaluate these methods, we have conducted four numerical tests with different problems. These tests illustrated the capabilities of our proposed models in solving possible real applications. In particular, the CP model outperformed the Metaheuristic algorithm in solving extra-large problems. These tests proved that path selection and operation sequencing are crucial in decreasing the makespan. Numerical results showed that the average improvement (decreasing) in makespan was up to 13% in large-scale problems when more candidate paths were available for each job.

6.1 Conclusion

In conclusion, our research makes three contributions. First, we formally define the MHNSP using a mathematical model that describes the complex structure and intricate material movements in the material handling network. In addition, we present the problem generation procedure that can construct arbitrary networks and generate jobs. This procedure involves constructing a random

connected graph, generating valid jobs along with their candidate paths, and saving the generated problems in structured and formatted files.



Second, we propose CP, IP, and DE models for solving the MHNSP. We design a hierarchical structure for the CP model to efficiently define the constraints on transportation jobs, candidate paths, and operations. The main contribution of the proposed IP model is that we ingeniously model the site buffer by identifying the operation overlapping condition using pair-wise logic relationships. In the DE model, we provide an efficient decoding procedure. This procedure adopts the discrete-event simulation technique to deal with events that set the start/end time of each operation.

Third, the proposed models were thoroughly and rigorously evaluated via four numerical tests. In the DE parameter-tuning test, we adopted the Taguchi method to identify the best parameter levels for each problem type. In the model performance comparison test, each model was applied to solve 360 randomly generated problems to compare their performances. The result showed that the IP model is suitable for small problems, and for large-scale problems, one should resort to the CP and the DE models, which have comparable performance. This test also showed that the average makespan decreases around 13% in large-scale problems when more candidate paths are considered for each job. Next, we tested the performance of the CP and the DE models in extra-large problems. The results showed that the CP model is more consistent and effective in finding better solutions than the DE model when having a generous solving time. Finally, the identical request test demonstrated the importance of path selection in improving

the makespan. In this test, we showed that a proper path selection can lead to a 35.7% decrease in makespan. This improvement mainly resulted from a more balanced workload among nodes and a shorter period of wait time spent in the transfer sites.

6.2 Future Work

Future work can be divided into two perspectives: improving the proposed models and exploring broader topics to enhance the manufacturing system.

6.2.1 Improvements in Modeling Techniques

CP Model

1. **Hierarchical Structure:** The current hierarchical structure discards scheduled operations when selecting a new path, leading to inefficiencies. Adopting a tree structure, similar to the depth-first search algorithm, can preserve common operations from different paths and reduce the number of interval variables used by the CP solver. This approach focuses on handling only the unscheduled operations when selecting a new path.
2. **Timeout Setting:** Defining a metric to balance solution quality and solving time can provide insights into setting appropriate timeouts. This is important because the CP model can yield good solutions if given sufficient solving time, as observed in the extra-large problem test.



IP Model

1. **Disjunctive Model Limitations:** The current IP model, based on the disjunctive approach [14, 17], requires establishing pair-wise overlapping relationships to model site capacity limits. This leads to exponential growth in constraints due to the lack of a “timeline” concept. Switching to a time-indexed model [13] could mitigate this issue and enhance performance despite its generally lower performance in job-shop scheduling problems.

DE Model

1. **Encoding Scheme:** Currently, a single integer array encodes all operations and decisions for path selection and operation sequencing. This can make the array excessively long for extra-large problems and hinder evolution since decisions are coupled. Separating operations and decisions into two arrays and evolving them individually can expedite solving speed and improve overall efficiency.

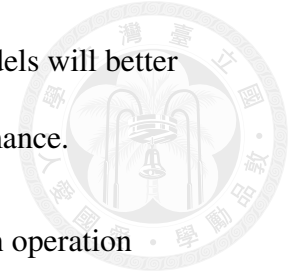
6.2.2 Improvements in Research Problems

To enhance the manufacturing system, future research should:

1. **Include Manufacturing Process:** Current models only consider material handling. Including manufacturing processes is crucial as some machines have specific requirements for material delivery, such as needing two material loads before starting or restrictions on the maximum hiatus between

deliveries. Considering these factors in optimization models will better reflect real-world conditions and improve system performance.

2. **Optimize Layout Design:** Combining layout design with operation scheduling is essential. These two aspects are highly interrelated, as identifying bottlenecks in the material handling network depends on having a proper schedule, and vice versa. A two-stage optimization approach, with the first stage focused on layout optimization and the second on MHNSP, can significantly enhance the manufacturing process.






Reference



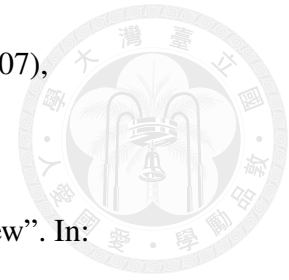
- [1] Hsu-Hsing Chen and Feng-Cheng Yang. “A Comprehensive Survey of Metaheuristic Algorithms Applying in Mechanical Design Optimization Problems”. In: *Proceedings of the International Symposium on Semiconductor Manufacturing Intelligence (ISMI 2022)*. Kinmen, Taiwan, 2022, pp. 11–13.
- [2] Sanchoy K Das and Prashanth Nagendra. “Selection of routes in a flexible manufacturing facility”. In: *International journal of production economics* 48.3 (1997), pp. 237–247.
- [3] Ayşe Tuğba Dosdoğru, Mustafa Göçken, and Faruk Geyik. “Integration of genetic algorithm and Monte Carlo to analyze the effect of routing flexibility”. In: *The International Journal of Advanced Manufacturing Technology* 81 (2015), pp. 1379–1389.
- [4] Ghada El Khayat, Andre Langevin, and Diane Riopel. “Integrated production and material handling scheduling using mathematical programming and constraint programming”. In: *European journal of operational research* 175.3 (2006), pp. 1818–1832.
- [5] Patrick-Oliver Groß et al. “Evaluation of alternative paths for reliable routing in city logistics”. In: *Transportation Research Procedia* 27 (2017), pp. 1195–1202.
- [6] Rongge Guo et al. “Time-dependent urban customized bus routing with path flexibility”. In: *IEEE Transactions on Intelligent Transportation Systems* 22.4 (2020), pp. 2381–2390.

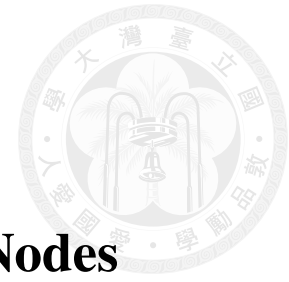


- [7] John H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. Ann Arbor, MI: U Michigan Press, 1975. URL: <https://psycnet.apa.org/record/1975-26618-000>.
- [8] Amir Hosseini, Alena Otto, and Erwin Pesch. “Scheduling in manufacturing with transportation: Classification and solution techniques”. In: *European Journal of Operational Research* (2023).
- [9] Yixiao Huang et al. “Time-dependent vehicle routing problem with path flexibility”. In: *Transportation Research Part B: Methodological* 95 (2017), pp. 169–195.
- [10] OA Joseph and R Sridharan. “Effects of routing flexibility, sequencing flexibility and scheduling decision rules on the performance of a flexible manufacturing system”. In: *The International Journal of Advanced Manufacturing Technology* 56 (2011), pp. 291–306.
- [11] Oya E Kara san, Mustafa C Pinar, and Hande Yaman. “The robust shortest path problem with interval data”. In: (2003).
- [12] Damla Kizilay, Pascal Van Hentenryck, and Deniz T Eliiyi. “Constraint programming models for integrated container terminal operations”. In: *European Journal of Operational Research* 286.3 (2020), pp. 945–962.
- [13] Emilia Kondili, Constantinos C Pantelides, and Roger WH Sargent. “A general algorithm for short-term scheduling of batch operations—I. MILP formulation”. In: *Computers & Chemical Engineering* 17.2 (1993), pp. 211–227.

- 
- [14] Ching-Jong Liao and Chii-Tsuen You. “An improved formulation for the job-shop scheduling problem”. In: *Journal of the Operational Research Society* 43.11 (1992), pp. 1047–1054.
- [15] Yi-Da Lin. “Fixed Material Transportation Network Job Scheduling Problem with Mathematical Programming Model and Heuristic Solving Methods”. MA thesis. National Taiwan University, 2023. DOI: [10.6342/NTU202302192](https://doi.org/10.6342/NTU202302192). URL: <http://tdr.lib.ntu.edu.tw/jspui/handle/123456789/88394>.
- [16] Yun-Yuan Liu. “Solving Flexible Job and Material Delivery Scheduling Problems Using Constraint Programming”. MA thesis. National Taiwan University, 2020. DOI: [10.6342/NTU202002054](https://doi.org/10.6342/NTU202002054). URL: <http://tdr.lib.ntu.edu.tw/jspui/handle/123456789/8362>.
- [17] Alan S Manne. “On the job-shop scheduling problem”. In: *Operations research* 8.2 (1960), pp. 219–223.
- [18] Andrea Rossi and Gino Dini. “An evolutionary approach to complex job-shop and flexible manufacturing system scheduling”. In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 215.2 (2001), pp. 233–245.
- [19] Andrea Rossi and Gino Dini. “Dynamic scheduling of FMS using a real-time genetic algorithm”. In: *International Journal of Production Research* 38.1 (2000), pp. 1–20.
- [20] Andrea Rossi and Gino Dini. “Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method”.

- In: *Robotics and Computer-Integrated Manufacturing* 23.5 (2007), pp. 503–516.
- [21] Roger L Sisson. “Methods of sequencing in job shops—a review”. In: *Operations Research* 7.1 (1959), pp. 10–29.
- [22] Rainer Storn and Kenneth Price. In: *Journal of Global Optimization* 11.4 (1997), pp. 341–359. ISSN: 0925-5001. DOI: [10.1023/a:1008202821328](https://doi.org/10.1023/a:1008202821328). URL: <http://dx.doi.org/10.1023/A:1008202821328>.
- [23] G Taguchi. *Introduction to quality engineering: designing quality into products and processes*. trid.trb.org, 1986. URL: <https://trid.trb.org/View/1179550>.
- [24] Hitoshi Tsubone and Mitsuyoshi Horikawa. “A comparison between machine flexibility and routing flexibility”. In: *International Journal of Flexible Manufacturing Systems* 11 (1999), pp. 83–101.
- [25] Pascal Van Hentenryck et al. “Constraint programming in OPL”. In: *International Conference on Principles and Practice of Declarative Programming*. Springer. 1999, pp. 98–116.
- [26] Hegen Xiong et al. “A survey of job shop scheduling problem: The types and models”. In: *Computers & Operations Research* 142 (2022), p. 105731.





Appendix A

Different Type of Transportation Nodes

To generalized the MHNSP, we can categorize the transportation node into two types: **vehicle node** and **conveyor node**. The vehicle node can be further divided into four types: *single-vehicle node*, *multi-vehicle node*, *single-capacitated vehicle node*, and *multi-capacitated vehicle node*, from the most simplified case to the most generalized case, according to their unit load capacity and the number of transportation equipment (vehicles) serving in the node.

In the vehicle node, the vehicle would move from a P/D point to another P/D point; therefore, the vehicle might not be able to immediately proceed with the next operation if the source P/D point of the next operation is not the same as the last stopping P/D point. We call this the **empty-car moving time** as it does not carry any payload when moving to the next source P/D point. This is often referred to as the sequence-dependent setup time in the literature.

In the conveyor node, there are generally only two P/D points, and all material processed by the node must move unidirectionally from one P/D point to the other, leading to an infinite traveling time in the opposite direction. On the other hand, there is no empty-car moving time in the conveyor node, but instead, the minimum interval between the entering time of one operation and the next is restricted.



We will discuss the details of each type of node in the following.

1. **Single-vehicle Node** (# of vehicle = 1, capacity = 1)

In the simplest case, only one vehicle (transportation equipment) serves in the node and possesses one unit load capacity. Therefore, the node can only process one operation at any moment.

2. **Multi-vehicle Node** (# of vehicle = α , capacity = 1)

In this case, there are α equivalent vehicles with one unit load capacity each in this node so that each operation can be processed by one and only one of the vehicles.

3. **Single-capacitated Vehicle Node** (# of vehicle = 1, capacity = β)

We consider the case that the vehicle has β unit load capacity so that the vehicle can complete β operations at the same time. In other words, before delivering the material to its target P/D point, the vehicle might stop at several midway P/D points to pick up additional materials. This is typically referred to as the vehicle routing problem.

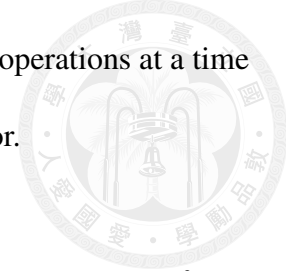
4. **Multi-capacitated Vehicle Node** (# of vehicle = α , capacity = β)

In the most generalized case, there are α vehicles, and each has β unit load capacity. Therefore, each operation can be processed by one of the vehicles which potentially has other operations in progress simultaneously.

5. **Conveyor Node** (capacity = β , entering interval = γ)

In a conveyor node, we do not consider the empty-car moving time, but there is a minimum interval γ between one operation's entering time and the next's

entering time. Moreover, the node can process at most β operations at a time due to the physical limitation of the length of the conveyor.



A.1 CP Model for Different Types of Transportation Nodes

For the node d , we can aggregate all the operations that will be potentially executed in the node in the set $\mathcal{P}^{(d)}$. Assume that there are \bar{p} operations in $\mathcal{P}^{(d)}$, we can locally index them as

$$\mathcal{P}^{(d)} = \{\rho_i \mid \rho_i = (\eta_i, \theta_i, \delta_i), i = 1, 2, \dots, \bar{p}\},$$

and the corresponding set of interval variables can be denoted as

$$\Gamma^{(d)} = \{t_i \mid i = 1, 2, \dots, \bar{p}\}$$

Single-vehicle Node (# of vehicle = 1, capacity = 1)

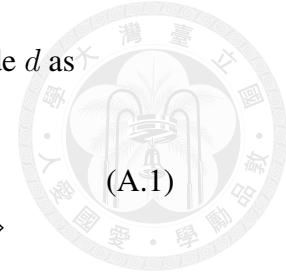
First, we construct the pair-wise from-to matrix $\tilde{M}^{(d)}$ to calculate the required empty-car moving time for any pair of operations of the node

$$\tilde{M}^{(d)} = [M^{(d)}(\delta_j, \theta_{j'})]_{\bar{p} \times \bar{p}}, \quad j, j' = 1, 2, \dots, \bar{p}$$

Second, we define the sequence variable of the node, and the type of each interval variable is determined by their index in $\Gamma^{(d)}$

$$q^{(d)} = seq(\Gamma^{(d)}, \{1, 2, \dots, \bar{p}\})$$

Finally, each interval variable t_i should have a minimum size that equals the minimum transporting time $M^{(d)}(\theta_i, \delta_i)$.



Now, we can establish the constraints for the single-vehicle node d as

$$noOverlap(q^{(d)}, \tilde{M}^{(d)}) \wedge \bigwedge_{i=1}^{\bar{p}} \left\{ p(t_i) \Rightarrow \left\{ e(t_i) - s(t_i) \geq M^{(d)}(\theta_i, \delta_i) \right\} \right\} \quad (A.1)$$

Multi-vehicle Node (# of vehicle = α , capacity = 1)

First, we construct the pair-wise from-to matrix

$$\tilde{M}^{(d)} = [M^{(d)}(\delta_j, \theta_{j'})]_{\bar{p} \times \bar{p}}, \quad j, j' = 1, 2, \dots, \bar{p}$$

Since every operation can choose to be executed by one of the α vehicles, we establish a set of interval variables $\Gamma_l^{(d)}$ for each vehicle l

$$\Gamma_l^{(d)} = \{t_i^{(l)} \mid opt_i^{(l)} = true, i = 1, 2, \dots, \bar{p}\}, \quad l = 1, 2, \dots, \alpha$$

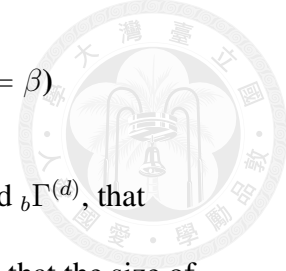
, and the associated sequence variables for each vehicle l

$$q_l^{(d)} = seq(\Gamma_l^{(d)}, \{1, 2, \dots, \bar{p}\}), \quad l = 1, 2, \dots, \alpha$$

Finally, each interval variable t_i should have a minimum size that equals the minimum transporting time $M^{(d)}(\theta_i, \delta_i)$.

We can establish the constraints for the multi-vehicle node d as

$$\bigwedge_{i=1}^{\bar{p}} alternative(t_i, \{t_i^{(l)} \mid l = 1, 2, \dots, \alpha\}) \wedge \bigwedge_{l=1}^{\alpha} noOverlap(q_l^{(d)}, \tilde{M}^{(d)}) \wedge \bigwedge_{i=1}^{\bar{p}} \left\{ p(t_i) \Rightarrow \left\{ e(t_i) - s(t_i) \geq M^{(d)}(\theta_i, \delta_i) \right\} \right\} \quad (A.2)$$



Single-capacitated Vehicle Node (# of vehicle = 1, capacity = β)

First, we construct two sets of interval variables, ${}_a\Gamma^{(d)}$ and ${}_b\Gamma^{(d)}$, that represent the picking and delivery processes, respectively. Note that the size of every interval variable in the two sets equals 0.

$${}_a\Gamma^{(d)} = \{{}_at_i \mid {}_aopt_i = true, i = 1, 2, \dots, \bar{p}\} \quad \# \text{ picking}$$

$${}_b\Gamma^{(d)} = \{{}_bt_i \mid {}_bopt_i = true, i = 1, 2, \dots, \bar{p}\} \quad \# \text{ delivery}$$

The sequence variable of the node can then be defined from ${}_a\Gamma^{(d)}$ and ${}_b\Gamma^{(d)}$, and the type of each interval is the P/D point it stops at.

$$q^{(d)} = seq({}_a\Gamma^{(d)} \cup {}_b\Gamma^{(d)}, \{\theta_1, \theta_2, \dots, \theta_{\bar{p}}\} \cup \{\delta_1, \delta_2, \dots, \delta_{\bar{p}}\})$$

To model the usage of the unit load capacity, we define the resource function $C^{(d)}$ by all operation interval variables in the node such that whenever an operation is present, it will cost one unit load capacity.

$$C^{(d)} = \sum_{i=1}^{\bar{p}} pulse(t_i, 1)$$

We can summarize the constraints for the single-capacitated vehicle node d



as

$$\begin{aligned}
 & noOverlap(q^{(d)}, M^{(d)}) \wedge \\
 & \bigwedge_{i=1}^{\bar{p}} span(t_i, \{at_i, bt_i\}) \wedge \\
 & \bigwedge_{i=1}^{\bar{p}} endBeforeStart(at_i, bt_i) \wedge \\
 & \bigwedge_{i=1}^{\bar{p}} alwaysIn(C^{(d)}, t_i, 0, \beta) \wedge \\
 & \bigwedge_{i=1}^{\bar{p}} \{p(t_i) \Rightarrow \{e(at_i) = s(at_i)\}\} \wedge \\
 & \bigwedge_{i=1}^{\bar{p}} \{p(t_i) \Rightarrow \{e(bt_i) = s(bt_i)\}\}
 \end{aligned} \tag{A.3}$$

Multi-capacitated Vehicle Node (# of vehicle = α , capacity = β)

First, we establish a set of interval variables $\Gamma_l^{(d)}$ for each vehicle l

$$\Gamma_l^{(d)} = \{t_i^{(l)} \mid opt_i^{(l)} = true, i = 1, 2, \dots, \bar{p}\}, \quad l = 1, 2, \dots, \alpha$$

Second, we define the picking and delivery set of interval variables, ${}_a\Gamma_l^{(d)}$ and ${}_b\Gamma_l^{(d)}$, for each vehicle l

$${}_a\Gamma_l^{(d)} = \{at_i^{(l)} \mid {}_aopt_i^{(l)} = true, i = 1, 2, \dots, \bar{p}\}, \quad l = 1, 2, \dots, \alpha \quad \# \text{ picking}$$

$${}_b\Gamma_l^{(d)} = \{bt_i^{(l)} \mid {}_bopt_i^{(l)} = true, i = 1, 2, \dots, \bar{p}\}, \quad l = 1, 2, \dots, \alpha \quad \# \text{ delivery}$$

, and the corresponding sequence variable for vehicle l can be defined as

$$q_l^{(d)} = seq({}_a\Gamma_l^{(d)} \cup {}_b\Gamma_l^{(d)}, \{\theta_1, \theta_2, \dots, \theta_{\bar{p}}\} \cup \{\delta_1, \delta_2, \dots, \delta_{\bar{p}}\}), \quad l = 1, 2, \dots, \alpha$$

Third, we have the resource function $C_l^{(d)}$ for each vehicle l

$$C_l^{(d)} = \sum_{i=1}^{\bar{p}} pulse(t_i^{(l)}, 1), \quad l = 1, 2, \dots, \alpha$$

Finally, we can derive the constraint for the multi-capacitated vehicle node d

$$\begin{aligned}
 & \bigwedge_{i=1}^{\bar{p}} \text{alternative}(t_i, \{t_i^{(l)} \mid l = 1, 2, \dots, \alpha\}) \wedge \\
 & \bigwedge_{l=1}^{\alpha} \text{noOverlap}(q_l^{(d)}, M^{(d)}) \wedge \\
 & \bigwedge_{l=1}^{\alpha} \bigwedge_{i=1}^{\bar{p}} \text{span}(t_i^{(l)}, \{at_i^{(l)}, bt_i^{(l)}\}) \wedge \\
 & \bigwedge_{l=1}^{\alpha} \bigwedge_{i=1}^{\bar{p}} \text{endBeforeStart}(at_i^{(l)}, bt_i^{(l)}) \wedge \\
 & \bigwedge_{l=1}^{\alpha} \bigwedge_{i=1}^{\bar{p}} \text{alwaysIn}(C_l^{(d)}, t_i^{(l)}, 0, \beta) \wedge \\
 & \bigwedge_{l=1}^{\alpha} \bigwedge_{i=1}^{\bar{p}} \{p(t_i^{(l)}) \Rightarrow \{e(at_i^{(l)}) = s(at_i^{(l)})\}\} \wedge \\
 & \bigwedge_{l=1}^{\alpha} \bigwedge_{i=1}^{\bar{p}} \{p(t_i^{(l)}) \Rightarrow \{e(bt_i^{(l)}) = s(bt_i^{(l)})\}\}
 \end{aligned} \tag{A.4}$$

Conveyor Node (capacity = β , entering interval = γ)

First, we define the set of interval variables ${}_h\Gamma^{(d)}$ to control the interval between two material entries.

$${}_h\Gamma^{(d)} = \{{}_ht_i \mid {}_hopt_i = true, i = 1, 2, \dots, \bar{p}\}$$

, and the corresponding sequence variable $q^{(d)} = seq({}_h\Gamma^{(d)})$

Second, the resource function $C^{(d)}$ can be written as

$$C^{(d)} = \sum_{i=1}^{\bar{p}} \text{pulse}(t_i, 1)$$

Now we have the constraint for the conveyor node d

$$\begin{aligned}
 & noOverlap(q^{(d)}) \wedge \\
 & \bigwedge_{i=1}^{\bar{p}} startAtStart(t_i, ht_i) \wedge \\
 & \bigwedge_{i=1}^{\bar{p}} \left\{ p(t_i) \Rightarrow \{e(ht_i) - s(ht_i) = \gamma\} \right\} \wedge \\
 & \bigwedge_{i=1}^{\bar{p}} alwaysIn(C^{(d)}, t_i, 0, \beta) \wedge \\
 & \bigwedge_{i=1}^{\bar{p}} \left\{ p(t_i) \Rightarrow \{e(t_i) - s(t_i) \geq M^{(d)}(\theta_i, \delta_i)\} \right\}
 \end{aligned} \tag{A.5}$$





Appendix B

Numerical Result Raw Data

Table B-1. Taguchi results of DE parameter-tuning.

Scale	Type	Exp.	Parameter level			Parameter level value			Run					SNR
			N	\tilde{M}	\tilde{C}	N	\tilde{M}	\tilde{C}	Run 1	Run 2	Run 3	Run 4	Run 5	
Small	$1/\infty$	1	1	1	1	8	0.001	0.050	95	95	95	95	95	-39.554
		2	1	2	2	8	0.010	0.100	95	95	95	95	95	-39.554
		3	1	3	3	8	0.100	0.200	95	95	95	95	95	-39.554
		4	2	1	2	64	0.001	0.100	95	95	95	95	95	-39.554
		5	2	2	3	64	0.010	0.200	95	95	95	95	95	-39.554
		6	2	3	1	64	0.100	0.050	95	95	95	95	95	-39.554
		7	3	1	3	128	0.001	0.200	95	95	95	95	95	-39.554
		8	3	2	1	128	0.010	0.050	95	95	95	95	95	-39.554
		9	3	3	2	128	0.100	0.100	95	95	95	95	95	-39.554
Small	$1/n$	1	1	1	1	8	0.001	0.050	95	95	95	95	95	-39.554
		2	1	2	2	8	0.010	0.100	95	95	95	95	95	-39.554
		3	1	3	3	8	0.100	0.200	95	95	95	95	95	-39.554
		4	2	1	2	64	0.001	0.100	95	95	95	95	95	-39.554
		5	2	2	3	64	0.010	0.200	95	95	95	95	95	-39.554
		6	2	3	1	64	0.100	0.050	95	95	95	95	95	-39.554
		7	3	1	3	128	0.001	0.200	95	95	95	95	95	-39.554
		8	3	2	1	128	0.010	0.050	95	95	95	95	95	-39.554
		9	3	3	2	128	0.100	0.100	95	95	95	95	95	-39.554
Small	n/∞	1	1	1	1	8	0.001	0.050	95	95	95	95	95	-39.554
		2	1	2	2	8	0.010	0.100	95	95	95	95	95	-39.554
		3	1	3	3	8	0.100	0.200	95	95	95	95	95	-39.554
		4	2	1	2	64	0.001	0.100	95	95	95	95	95	-39.554
		5	2	2	3	64	0.010	0.200	95	95	95	95	95	-39.554
		6	2	3	1	64	0.100	0.050	95	95	95	95	95	-39.554
		7	3	1	3	128	0.001	0.200	95	95	95	95	95	-39.554
		8	3	2	1	128	0.010	0.050	95	95	95	95	95	-39.554
		9	3	3	2	128	0.100	0.100	95	95	95	95	95	-39.554
Small	n/n	1	1	1	1	8	0.001	0.050	95	95	95	95	95	-39.554
		2	1	2	2	8	0.010	0.100	95	95	95	95	95	-39.554
		3	1	3	3	8	0.100	0.200	95	95	95	95	95	-39.554
		4	2	1	2	64	0.001	0.100	95	95	95	95	95	-39.554
		5	2	2	3	64	0.010	0.200	95	95	95	95	95	-39.554
		6	2	3	1	64	0.100	0.050	95	95	95	95	95	-39.554
		7	3	1	3	128	0.001	0.200	95	95	95	95	95	-39.554
		8	3	2	1	128	0.010	0.050	95	95	95	95	95	-39.554
		9	3	3	2	128	0.100	0.100	95	95	95	95	95	-39.554

Continued on next page

Table B-1. Taguchi results of DE parameter tuning (cont.)

Scale	Type	Exp.	Parameter level			Parameter level value			Run					SNR
			N	\tilde{M}	\tilde{C}	N	\tilde{M}	\tilde{C}_r	Run 1	Run 2	Run 3	Run 4	Run 5	
Medium	$1/\infty$	1	1	1	1	8	0.001	0.050	220	220	220	220	220	-46.848
		2	1	2	2	8	0.010	0.100	220	220	220	220	220	-46.848
		3	1	3	3	8	0.100	0.200	220	220	220	220	220	-46.848
		4	2	1	2	64	0.001	0.100	220	220	220	220	220	-46.848
		5	2	2	3	64	0.010	0.200	220	220	220	220	220	-46.848
		6	2	3	1	64	0.100	0.050	220	220	220	220	220	-46.848
		7	3	1	3	128	0.001	0.200	220	220	220	220	220	-46.848
		8	3	2	1	128	0.010	0.050	220	220	220	220	220	-46.848
		9	3	3	2	128	0.100	0.100	220	220	220	220	220	-46.848
	$1/n$	1	1	1	1	8	0.001	0.050	220	220	220	220	220	-46.848
		2	1	2	2	8	0.010	0.100	220	220	220	220	220	-46.848
		3	1	3	3	8	0.100	0.200	220	220	220	220	220	-46.848
		4	2	1	2	64	0.001	0.100	220	220	220	220	220	-46.848
		5	2	2	3	64	0.010	0.200	220	220	220	220	220	-46.848
		6	2	3	1	64	0.100	0.050	220	220	220	220	220	-46.848
		7	3	1	3	128	0.001	0.200	220	220	220	220	220	-46.848
		8	3	2	1	128	0.010	0.050	220	220	220	220	220	-46.848
		9	3	3	2	128	0.100	0.100	220	220	220	220	220	-46.848
Medium	n/∞	1	1	1	1	8	0.001	0.050	207	207	207	207	207	-46.319
		2	1	2	2	8	0.010	0.100	207	207	207	207	207	-46.319
		3	1	3	3	8	0.100	0.200	207	207	207	207	207	-46.319
		4	2	1	2	64	0.001	0.100	207	207	207	207	207	-46.319
		5	2	2	3	64	0.010	0.200	207	207	207	207	207	-46.319
		6	2	3	1	64	0.100	0.050	207	207	207	207	207	-46.319
		7	3	1	3	128	0.001	0.200	207	207	207	207	207	-46.319
		8	3	2	1	128	0.010	0.050	207	207	207	207	207	-46.319
		9	3	3	2	128	0.100	0.100	207	207	207	207	207	-46.319
	n/n	1	1	1	1	8	0.001	0.050	207	207	207	207	207	-46.319
		2	1	2	2	8	0.010	0.100	207	207	207	207	207	-46.319
		3	1	3	3	8	0.100	0.200	207	207	207	207	207	-46.319
		4	2	1	2	64	0.001	0.100	207	207	207	207	207	-46.319
		5	2	2	3	64	0.010	0.200	207	207	207	207	207	-46.319
		6	2	3	1	64	0.100	0.050	207	207	207	207	207	-46.319
		7	3	1	3	128	0.001	0.200	207	207	207	207	207	-46.319
		8	3	2	1	128	0.010	0.050	207	207	207	207	207	-46.319
		9	3	3	2	128	0.100	0.100	207	207	207	207	207	-46.319

Continued on next page

Table B-1. Taguchi results of DE parameter tuning (cont.)

Scale	Type	Exp.	Parameter level			Parameter level value			Run					SNR
			N	\tilde{M}	\tilde{C}	N	\tilde{M}	\tilde{C}_r	Run 1	Run 2	Run 3	Run 4	Run 5	
Large	1/ ∞	1	1	1	1	8	0.001	0.050	172	171	172	171	172	-44.690
		2	1	2	2	8	0.010	0.100	171	171	171	171	171	-44.660
		3	1	3	3	8	0.100	0.200	171	171	171	171	171	-44.660
		4	2	1	2	64	0.001	0.100	171	171	171	171	171	-44.660
		5	2	2	3	64	0.010	0.200	171	171	171	171	171	-44.660
		6	2	3	1	64	0.100	0.050	171	171	171	171	171	-44.660
		7	3	1	3	128	0.001	0.200	171	171	171	171	171	-44.660
		8	3	2	1	128	0.010	0.050	171	171	171	171	171	-44.660
		9	3	3	2	128	0.100	0.100	171	171	171	171	171	-44.660
	1/ n	1	1	1	1	8	0.001	0.050	171	172	172	172	171	-44.690
		2	1	2	2	8	0.010	0.100	171	171	171	171	171	-44.660
		3	1	3	3	8	0.100	0.200	171	171	171	171	171	-44.660
		4	2	1	2	64	0.001	0.100	171	171	171	171	171	-44.660
		5	2	2	3	64	0.010	0.200	171	171	171	171	171	-44.660
		6	2	3	1	64	0.100	0.050	171	171	171	171	171	-44.660
		7	3	1	3	128	0.001	0.200	171	171	171	171	171	-44.660
		8	3	2	1	128	0.010	0.050	171	171	171	171	171	-44.660
		9	3	3	2	128	0.100	0.100	171	171	171	171	171	-44.660
	n/∞	1	1	1	1	8	0.001	0.050	203	207	221	209	213	-46.473
		2	1	2	2	8	0.010	0.100	184	193	187	186	197	-45.550
		3	1	3	3	8	0.100	0.200	204	215	194	222	219	-46.488
		4	2	1	2	64	0.001	0.100	210	213	222	212	214	-46.618
		5	2	2	3	64	0.010	0.200	225	220	227	222	213	-46.906
		6	2	3	1	64	0.100	0.050	231	201	214	229	197	-46.643
		7	3	1	3	128	0.001	0.200	221	233	212	237	217	-47.013
		8	3	2	1	128	0.010	0.050	232	234	227	214	227	-47.117
		9	3	3	2	128	0.100	0.100	223	225	228	235	234	-47.199
n/n	1	1	1	1	8	0.001	0.050	209	214	191	210	204	-46.267	
	2	1	2	2	8	0.010	0.100	214	179	197	211	189	-45.952	
	3	1	3	3	8	0.100	0.200	202	208	198	195	213	-46.163	
	4	2	1	2	64	0.001	0.100	228	229	229	222	220	-47.068	
	5	2	2	3	64	0.010	0.200	226	208	230	226	225	-46.971	
	6	2	3	1	64	0.100	0.050	223	208	237	239	223	-47.093	
	7	3	1	3	128	0.001	0.200	240	213	227	213	237	-47.093	
	8	3	2	1	128	0.010	0.050	234	240	227	226	253	-47.466	
	9	3	3	2	128	0.100	0.100	229	223	212	217	219	-46.851	

Table B-2. Objective values of model performance comparison test.

Scale	Type Net. ID	$1/\infty$							n/∞							$1/n$							n/n						
		CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5
	S-01	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95		
	S-02	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118	118		
	S-03	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123	123		
	S-04	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98		
	S-05	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102		
	S-06	122	122	122	122	122	122	122	122	122	122	122	122	122	122	122	122	122	122	122	122	122	122	122	122	122	122		
	S-07	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120		
	S-08	210	210	210	210	210	210	210	210	210	210	210	210	210	210	210	210	210	210	210	210	210	210	210	210	210	210		
	S-09	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105		
	S-10	107	107	107	107	107	107	107	107	107	108	107	107	107	107	107	107	107	107	107	107	108	108	107	108	108	108		
	S-11	103	103	103	103	103	103	103	103	103	103	105	105	103	103	103	103	103	103	103	103	103	103	103	103	103	103		
	S-12	139	139	139	139	139	139	139	118	118	118	118	118	118	139	139	139	139	139	139	139	118	118	118	118	118	118		
	S-13	136	136	136	136	136	136	136	135	135	135	135	135	135	136	136	136	136	136	136	135	135	136	136	135	135	135		
	S-14	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83		
Small	S-15	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106	106		
	S-16	72	72	72	72	72	72	72	72	72	72	72	72	72	72	72	72	72	72	72	72	72	72	72	72	72	72		
	S-17	162	162	162	162	162	162	162	120	120	120	120	120	120	162	162	162	162	162	162	120	120	120	120	120	120	120		
	S-18	108	108	108	108	108	108	108	108	108	108	108	108	108	108	108	108	108	108	108	108	108	108	108	108	108	108	108	
	S-19	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	116	
	S-20	121	121	121	121	121	121	121	118	118	118	118	121	119	118	121	121	121	121	121	121	118	118	121	118	118	118	121	
	S-21	98	98	98	98	98	98	98	98	98	98	98	98	98	98	99	99	99	99	99	99	99	99	99	99	100	100	99	
	S-22	117	117	117	118	117	117	117	107	107	107	107	107	107	107	117	117	117	117	117	117	107	107	107	107	107	107	107	
	S-23	65	65	65	65	65	65	65	65	65	65	65	65	65	65	66	66	66	66	66	66	66	66	66	66	66	66	66	
	S-24	120	120	120	120	120	120	120	115	115	115	115	115	115	115	120	120	120	120	120	120	115	115	115	115	115	115	115	
	S-25	70	-	70	70	70	70	70	70	70	70	70	70	70	70	-	70	70	70	70	70	70	70	70	70	70	70	70	
	S-26	83	83	83	83	83	83	83	83	83	87	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	83	
	S-27	196	196	196	196	196	196	196	169	169	169	169	169	169	169	196	196	196	196	196	196	169	169	169	169	169	169	169	
	S-28	110	110	110	110	110	110	110	102	102	107	102	102	105	102	112	112	114	112	112	112	102	102	102	102	105	102		
	S-29	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	
S-30	106	106	106	106	106	106	106	98	98	98	98	98	98	98	106	106	106	106	106	106	98	98	98	98	98	98	98		

Continued on next page

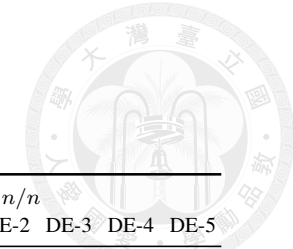


Table B-2. Objective values of model performance comparison test (cont.)

Scale	Type Net. ID	1/∞						n/∞						1/n						n/n									
		CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5
	M-01	220	220	220	220	221	220	220	207	208	207	207	207	207	207	220	220	220	220	225	220	220	207	207	207	207	207	207	
	M-02	140	140	140	140	140	140	140	132	132	139	139	139	140	139	140	140	140	140	140	140	140	132	160	140	139	140	139	132
	M-03	177	177	177	177	177	177	177	172	178	172	173	172	173	173	177	177	177	177	177	177	177	172	-	172	172	172	172	176
	M-04	180	180	180	180	180	180	180	178	-	183	184	180	180	178	180	180	180	180	180	180	180	178	178	180	180	180	178	178
	M-05	156	156	156	156	156	156	156	146	155	151	151	155	151	151	156	156	156	156	156	156	156	146	187	155	155	155	155	160
	M-06	202	202	202	202	202	202	202	184	184	185	185	185	185	185	202	202	202	202	202	202	202	184	184	185	185	185	185	185
	M-07	167	167	167	167	167	167	167	161	161	161	166	166	161	166	167	167	167	167	167	167	167	161	166	161	161	161	166	161
	M-08	144	144	144	144	144	144	144	119	119	122	119	119	119	122	144	144	144	144	144	144	144	119	119	121	122	119	120	120
	M-09	198	198	200	200	198	200	200	176	254	176	176	176	176	176	198	198	200	200	198	198	200	176	206	185	178	192	177	176
	M-10	189	189	190	189	189	189	189	188	188	188	188	188	188	188	189	189	189	189	189	189	189	188	188	188	188	188	188	188
	M-11	174	174	174	174	174	174	174	148	148	149	149	149	148	149	174	174	174	174	174	174	174	148	148	148	148	149	149	148
	M-12	147	147	147	147	147	147	147	145	145	145	145	145	145	145	147	147	147	147	147	147	147	145	145	145	145	145	145	145
	M-13	147	147	147	155	147	155	147	138	144	144	154	138	138	154	147	147	147	147	147	158	147	138	158	153	150	151	144	144
	M-14	195	195	195	195	195	195	195	166	177	184	181	166	177	187	195	195	195	195	195	195	166	190	166	179	178	177	179	179
	M-15	367	367	367	367	367	367	367	367	367	367	367	367	367	367	367	367	367	367	373	367	367	367	367	367	367	367	367	367
Medium	M-16	183	183	183	183	183	183	183	142	186	143	149	142	142	142	183	183	183	183	183	183	142	196	146	152	157	142	150	150
	M-17	253	253	253	253	253	253	253	183	187	183	183	187	183	183	253	253	253	253	253	253	187	206	199	199	199	199	199	199
	M-18	228	228	228	228	228	228	228	226	226	234	232	233	233	232	228	228	228	228	228	228	228	226	241	233	233	234	234	234
	M-19	178	178	178	186	181	178	186	178	178	178	178	178	178	178	178	178	186	178	178	178	178	178	178	178	178	178	178	178
	M-20	242	242	243	243	244	243	243	214	215	224	229	223	215	217	242	242	248	243	243	243	242	215	225	229	231	215	223	230
	M-21	180	180	180	180	180	180	180	162	162	168	170	168	162	162	180	180	180	180	180	180	180	162	170	168	162	168	168	162
	M-22	201	201	201	201	201	201	201	140	140	140	140	140	140	140	201	201	201	201	201	201	140	140	140	140	140	140	140	140
	M-23	172	172	172	172	172	172	172	154	-	155	154	154	154	154	172	172	172	172	172	172	154	157	154	154	154	154	154	154
	M-24	278	278	278	278	278	278	278	278	385	278	278	278	278	278	278	278	278	278	278	278	278	-	278	278	278	278	278	278
	M-25	114	114	115	115	114	115	114	112	112	123	123	123	123	123	114	114	115	114	122	114	114	112	112	123	123	112	123	112
	M-26	114	114	114	114	114	114	114	114	114	114	114	114	114	114	117	117	117	117	117	117	114	114	114	114	114	114	114	114
	M-27	178	178	178	178	178	178	178	159	159	159	159	159	159	159	178	178	178	178	178	202	178	159	159	159	159	159	159	159
	M-28	178	178	178	178	178	178	178	151	157	153	158	154	154	154	178	178	178	178	178	178	151	184	154	158	154	154	154	153
	M-29	248	248	248	248	248	248	248	248	275	248	248	248	252	248	248	248	248	248	248	248	248	-	248	252	263	248	248	248
	M-30	210	210	210	210	210	210	210	154	154	154	154	154	154	154	210	210	210	210	210	210	154	154	154	154	154	154	154	154

Continued on next page

Table B-2. Objective values of model performance comparison test (cont.)

Scale	Type Net. ID	$1/\infty$							n/∞							$1/n$							n/n						
		CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5
Large	L-01	171	171	171	171	171	171	171	138	272	138	157	138	138	155	171	171	171	171	171	171	171	138	-	145	146	155	160	138
	L-02	210	210	210	210	210	210	210	198	-	198	198	210	204	202	210	210	210	210	210	210	210	216	-	210	213	203	210	202
	L-03	171	171	171	171	171	171	171	191	3600	167	180	167	180	167	171	171	171	171	171	171	171	180	-	167	167	180	175	180
	L-04	222	222	222	222	222	222	222	231	-	197	200	207	207	197	222	222	222	222	222	222	222	197	-	197	197	200	213	197
	L-05	180	180	180	180	180	180	180	163	-	163	173	166	163	178	180	180	180	180	180	180	180	163	-	173	166	166	168	170
	L-06	118	118	118	118	118	118	118	118	165	118	118	118	118	118	118	118	118	118	118	118	118	118	159	118	125	118	118	118
	L-07	143	143	143	143	143	143	143	131	-	131	131	131	131	131	143	143	143	143	143	143	143	131	180	131	131	131	131	131
	L-08	224	224	224	224	224	224	224	154	3180	154	154	154	154	154	224	224	224	224	224	224	224	154	-	154	163	155	167	154
	L-09	234	234	234	234	234	234	234	203	235	202	203	199	204	201	234	234	234	234	234	234	234	203	-	203	198	203	198	205
	L-10	171	171	171	171	171	171	171	151	151	151	151	151	155	151	171	171	171	171	171	171	171	151	151	151	159	159	151	155
	L-11	188	188	188	188	188	188	188	171	302	173	173	180	173	183	188	188	188	188	188	188	188	171	436	173	173	173	180	175
	L-12	269	269	269	269	269	269	269	239	311	233	235	233	233	235	269	269	269	269	269	269	269	233	-	236	233	235	235	235
	L-13	202	202	202	202	202	202	202	180	346	195	185	189	192	189	202	202	202	202	202	202	202	192	-	210	193	205	189	204
	L-14	232	232	232	232	232	232	232	246	-	193	195	193	200	193	232	232	232	232	232	232	232	213	-	216	202	202	195	195
	L-15	149	149	149	149	149	149	149	125	125	125	125	125	125	125	149	149	149	149	149	149	149	125	146	126	125	125	126	130
	L-16	200	200	200	200	200	200	200	170	-	171	179	175	171	175	200	200	200	200	200	200	200	168	235	182	171	179	180	184
	L-17	230	230	230	230	230	230	230	140	256	143	140	140	140	149	230	230	230	230	230	230	230	140	278	142	140	143	142	142
	L-18	126	126	126	126	126	126	126	106	106	106	106	106	106	106	126	126	126	126	126	126	126	106	106	106	106	106	106	106
	L-19	192	192	192	192	192	192	192	153	213	168	168	162	168	163	192	192	192	192	192	192	192	153	-	168	165	173	177	165
	L-20	214	214	214	214	214	214	214	195	253	194	194	195	194	195	214	214	214	214	214	214	214	197	-	194	195	194	194	194
	L-21	264	264	264	264	264	264	264	253	-	257	261	270	257	262	264	264	264	264	264	264	264	256	-	267	257	252	260	264
	L-22	196	196	196	196	196	196	196	155	157	163	155	155	155	155	196	196	196	196	196	196	196	155	155	163	155	155	155	155
	L-23	177	177	177	177	177	177	177	153	217	160	160	153	153	150	177	177	177	177	177	177	177	149	-	153	155	153	153	153
	L-24	134	134	134	134	134	134	134	122	154	122	122	122	122	122	134	134	134	134	134	134	134	122	-	126	122	122	126	122
	L-25	129	129	129	129	129	129	129	116	-	116	116	116	116	116	129	129	129	129	129	129	129	116	116	116	116	116	116	116
	L-26	184	-	184	184	184	184	184	184	-	188	201	196	201	201	184	184	184	184	184	184	184	201	-	201	201	196	201	197
	L-27	178	178	178	178	178	178	178	158	228	166	166	158	166	167	178	178	178	178	178	178	178	158	208	166	172	158	172	173
	L-28	257	257	257	257	257	257	257	147	-	159	148	157	159	159	257	257	257	257	257	257	257	163	-	159	162	158	161	159
	L-29	197	197	197	197	197	197	197	184	193	193	184	184	184	184	197	197	197	197	197	197	197	184	299	184	184	184	184	193
	L-30	214	214	214	214	214	214	214	199	257	193	189	187	193	189	214	214	214	214	214	214	214	197	-	195	193	193	190	193

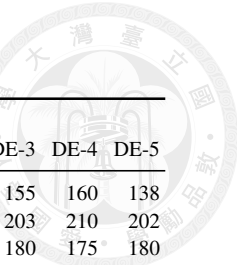




Table B-3. Solving time of model performance comparison test.

Scale	Type Net. ID	$1/\infty$							n/∞							$1/n$							n/n						
		CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5
Small	S-01	0.06	0.02	2.00	2.00	2.00	2.00	2.00	0.11	0.03	20.00	20.00	20.00	20.00	20.00	0.04	0.05	2.00	2.00	2.00	2.00	2.00	0.12	0.08	20.00	20.00	20.00	20.00	20.00
	S-02	0.53	0.03	2.00	2.00	2.00	2.00	2.00	0.68	0.16	20.00	20.00	20.00	20.00	20.00	0.51	0.03	2.00	2.00	2.00	2.00	2.00	0.99	0.22	20.00	20.00	20.00	20.00	20.00
	S-03	0.24	0.01	2.00	2.00	2.00	2.00	2.00	0.32	0.00	20.00	20.00	20.00	20.00	20.00	0.47	0.02	2.00	2.00	2.00	2.00	2.00	0.47	0.02	20.00	20.00	20.00	20.00	20.00
	S-04	0.24	0.00	2.00	2.00	2.00	2.00	2.00	0.24	0.00	20.00	20.00	20.00	20.00	20.00	0.45	0.02	2.00	2.00	2.00	2.00	2.00	0.45	0.03	20.00	20.00	20.00	20.00	20.00
	S-05	0.03	0.02	2.00	2.00	2.00	2.00	2.00	0.27	0.03	20.00	20.00	20.00	20.00	20.00	0.04	0.00	2.00	2.00	2.00	2.00	2.00	0.46	0.05	20.00	20.00	20.00	20.00	20.00
	S-06	0.34	0.00	2.00	2.00	2.00	2.00	2.00	0.54	0.06	20.00	20.00	20.00	20.00	20.00	0.56	0.02	2.00	2.00	2.00	2.00	2.00	0.63	0.14	20.00	20.00	20.00	20.00	20.00
	S-07	0.95	0.01	2.00	2.00	2.00	2.00	2.00	2.19	0.98	20.00	20.00	20.00	20.00	20.00	0.66	0.03	2.00	2.00	2.00	2.00	2.00	2.92	1.88	20.00	20.00	20.00	20.00	20.00
	S-08	1.01	0.05	2.00	2.00	2.00	2.00	2.00	1.62	0.94	20.00	20.00	20.00	20.00	20.00	1.09	0.16	2.00	2.00	2.00	2.00	2.00	1.78	2.55	20.00	20.00	20.00	20.00	20.00
	S-09	0.23	0.00	2.00	2.00	2.00	2.00	2.00	0.24	0.03	20.00	20.00	20.00	20.00	20.00	0.46	0.02	2.00	2.00	2.00	2.00	2.00	0.46	0.05	20.00	20.00	20.00	20.00	20.00
	S-10	0.42	0.02	2.00	2.00	2.00	2.00	2.00	0.95	0.44	20.00	20.00	20.00	20.00	20.00	0.55	0.03	2.00	2.00	2.00	2.00	2.00	0.99	0.27	20.00	20.00	20.00	20.00	20.00
	S-11	0.55	0.05	2.00	2.00	2.00	2.00	2.00	2.57	5.98	20.00	20.00	20.00	20.00	20.00	0.53	0.11	2.00	2.00	2.00	2.00	2.00	2.62	4.83	20.00	20.00	20.00	20.00	20.00
	S-12	0.23	0.02	2.00	2.00	2.00	2.00	2.00	0.65	0.08	20.00	20.00	20.00	20.00	20.00	0.46	0.02	2.00	2.00	2.00	2.00	2.00	0.71	0.14	20.00	20.00	20.00	20.00	20.00
	S-13	0.70	0.06	2.00	2.00	2.00	2.00	2.00	3.52	1.67	20.00	20.00	20.00	20.00	20.00	0.93	0.06	2.00	2.00	2.00	2.00	2.00	3.35	3.25	20.00	20.00	20.00	20.00	20.00
	S-14	0.04	0.02	2.00	2.00	2.00	2.00	2.00	0.04	0.03	20.00	20.00	20.00	20.00	20.00	0.04	0.02	2.00	2.00	2.00	2.00	2.00	0.04	0.03	20.00	20.00	20.00	20.00	20.00
	S-15	0.40	0.00	2.00	2.00	2.00	2.00	2.00	0.61	0.30	20.00	20.00	20.00	20.00	20.00	0.52	0.02	2.00	2.00	2.00	2.00	2.00	0.80	0.69	20.00	20.00	20.00	20.00	20.00
	S-16	0.03	0.02	2.00	2.00	2.00	2.00	2.00	0.03	0.00	20.00	20.00	20.00	20.00	20.00	0.01	0.02	2.00	2.00	2.00	2.00	2.00	0.01	0.00	20.00	20.00	20.00	20.00	20.00
	S-17	0.70	0.02	2.00	2.00	2.00	2.00	2.00	0.59	0.20	20.00	20.00	20.00	20.00	20.00	0.63	0.03	2.00	2.00	2.00	2.00	2.00	0.77	0.78	20.00	20.00	20.00	20.00	20.00
	S-18	0.33	0.02	2.00	2.00	2.00	2.00	2.00	0.35	0.02	20.00	20.00	20.00	20.00	20.00	0.49	0.05	2.00	2.00	2.00	2.00	2.00	0.49	0.05	20.00	20.00	20.00	20.00	20.00
	S-19	0.31	0.02	2.00	2.00	2.00	2.00	2.00	1.45	0.36	20.00	20.00	20.00	20.00	20.00	0.12	0.03	2.00	2.00	2.00	2.00	2.00	1.33	1.22	20.00	20.00	20.00	20.00	20.00
	S-20	0.53	0.05	2.00	2.00	2.00	2.00	2.00	0.90	0.53	20.00	20.00	20.00	20.00	20.00	0.98	0.05	2.00	2.00	2.00	2.00	2.00	0.99	1.03	20.00	20.00	20.11	20.00	20.00
	S-21	0.41	0.02	2.00	2.00	2.00	2.00	2.00	0.46	0.12	20.00	20.00	20.00	20.00	20.00	0.49	0.02	2.00	2.00	2.00	2.00	2.00	0.55	0.58	20.00	20.00	20.00	20.00	20.00
	S-22	0.56	0.03	2.00	2.00	2.00	2.00	2.00	2.36	1.98	20.00	20.00	20.00	20.00	20.00	0.76	0.03	2.00	2.00	2.00	2.00	2.00	2.02	5.28	20.00	20.00	20.00	20.00	20.00
	S-23	0.08	0.02	2.00	2.00	2.00	2.00	2.00	0.10	0.02	20.00	20.00	20.00	20.00	20.00	0.10	0.02	2.00	2.00	2.00	2.00	2.00	0.08	0.03	20.00	20.00	20.00	20.00	20.00
	S-24	0.40	0.03	2.00	2.00	2.00	2.00	2.00	0.49	0.45	20.00	20.00	20.00	20.00	20.00	0.42	0.03	2.00	2.00	2.00	2.00	2.00	0.60	0.48	20.00	20.00	20.00	20.00	20.00
	S-25	0.14	-	2.00	2.00	2.00	2.00	2.00	0.15	0.02	20.00	20.00	20.00	20.00	20.00	0.38	-	2.00	2.00	2.00	2.00	2.00	0.39	0.02	20.00	20.00	20.00	20.00	20.00
	S-26	0.20	0.01	2.00	2.00	2.00	2.00	2.00	0.28	0.05	20.00	20.10	20.00	20.00	20.00	0.38	0.01	2.00	2.00	2.00	2.00	2.00	0.38	0.09	20.00	20.00	20.00	20.00	20.00
	S-27	0.73	0.03	2.00	2.00	2.00	2.00	2.00	0.53	0.34	20.00	20.00	20.00	20.00	20.00	0.92	0.05	2.00	2.00	2.00	2.00	2.00	0.75	0.48	20.00	20.00	20.00	20.00	20.00
	S-28	0.55	0.02	2.00	2.00	2.00	2.00	2.00	1.20	0.64	20.00	20.00	20.00	20.00	20.00	0.59	0.03	2.00	2.00	2.00	2.00	2.00	1.62	1.08	20.00	20.00	20.00	20.00	20.00
	S-29	0.15	0.02	2.00	2.00	2.00	2.00	2.00	0.49	0.02	20.00	20.00	20.00	20.00	20.00	0.42	0.00	2.00	2.00	2.00	2.00	2.00	0.52	0.05	20.00	20.00	20.00	20.00	20.00
	S-30	0.06	0.01	2.00	2.00	2.00	2.00	2.00	0.38	0.06	20.00	20.00	20.00	20.00	20.00	0.08	0.02	2.00	2.00	2.00	2.00	2.00	0.54	0.09	20.00	20.00	20.00	20.00	20.00

Continued on next page

Numerical Result Raw Data

Table B-3. Solving time of model performance comparison test (cont.)

Scale	Type Net. ID	1/∞							n/∞							1/n							n/n							
		CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	
	M-01	1.77	0.28	2.00	2.00	2.00	2.00	2.00	21.16	60.03	20.00	20.00	20.00	20.00	20.00	1.45	0.17	2.00	2.00	2.00	2.00	2.00	2.00	60.02	60.03	20.00	20.00	20.00	20.00	20.00
	M-02	0.39	0.03	2.00	2.00	2.00	2.00	2.00	7.56	15.69	20.00	20.00	20.00	20.00	20.00	0.55	0.03	2.00	2.00	2.00	2.00	2.00	2.00	4.89	60.06	20.00	20.00	20.00	20.00	20.00
	M-03	1.10	0.03	2.00	2.00	2.00	2.00	2.00	60.13	60.17	20.00	20.00	20.00	20.00	20.00	0.90	0.03	2.00	2.00	2.00	2.00	2.00	2.00	60.14	-	20.00	20.00	20.00	20.00	20.00
	M-04	0.67	0.08	2.00	2.00	2.00	2.00	2.00	1.98	-	20.00	20.00	20.00	20.00	20.00	0.81	0.11	2.00	2.00	2.00	2.00	2.00	2.00	1.70	1.11	20.00	20.00	20.00	20.00	20.00
	M-05	1.44	0.16	2.00	2.00	2.00	2.00	2.00	29.05	60.05	20.00	20.00	20.00	20.00	20.00	1.26	0.09	2.00	2.00	2.00	2.00	2.00	2.00	10.07	60.08	20.00	20.00	20.00	20.00	20.00
	M-06	0.86	0.05	2.00	2.00	2.00	2.00	2.00	3.15	4.58	20.00	20.00	20.00	20.00	20.00	0.90	0.06	2.00	2.00	2.00	2.00	2.00	2.00	4.19	11.72	20.00	20.00	20.00	20.00	20.00
	M-07	0.58	0.02	2.00	2.00	2.00	2.00	2.00	8.17	14.03	20.00	20.00	20.00	20.00	20.00	0.67	0.02	2.00	2.00	2.00	2.00	2.00	2.00	5.88	60.14	20.00	20.00	20.00	20.00	20.00
	M-08	0.76	0.05	2.00	2.00	2.00	2.00	2.00	6.84	15.36	20.00	20.00	20.00	20.00	20.00	1.21	0.03	2.00	2.00	2.00	2.00	2.00	2.00	7.27	60.03	20.00	20.00	20.00	20.00	20.00
	M-09	1.48	0.11	2.00	2.00	2.00	2.00	2.00	37.91	60.09	20.00	20.00	20.00	20.00	20.00	1.52	0.16	2.00	2.00	2.00	2.00	2.00	2.00	35.61	60.16	20.00	20.00	20.00	20.00	20.00
	M-10	1.42	0.12	2.00	2.00	2.00	2.00	2.12	17.13	19.95	20.00	20.00	20.00	20.00	20.00	1.77	0.45	2.00	2.00	2.00	2.00	2.00	2.00	10.22	38.77	20.00	20.00	20.00	20.00	20.00
	M-11	0.87	0.03	2.00	2.00	2.00	2.00	2.00	5.08	12.91	20.00	20.00	20.00	20.00	20.00	1.04	0.02	2.00	2.00	2.00	2.00	2.00	2.00	6.99	60.06	20.00	20.00	20.00	20.00	20.00
	M-12	1.06	0.05	2.00	2.00	2.00	2.00	2.00	2.54	2.09	20.00	20.00	20.00	20.00	20.00	0.66	0.06	2.00	2.00	2.00	2.00	2.00	2.00	2.56	5.76	20.00	20.00	20.00	20.00	20.00
	M-13	0.80	0.06	2.00	2.00	2.00	2.00	2.00	6.72	60.03	20.00	20.00	20.00	20.00	20.00	0.90	0.16	2.00	2.00	2.00	2.00	2.00	2.00	12.88	60.03	20.00	20.00	20.00	20.00	20.00
	M-14	1.08	0.11	2.00	2.00	2.00	2.00	2.00	54.65	60.06	20.00	20.00	20.00	20.00	20.00	1.33	0.14	2.00	2.00	2.13	2.00	2.00	2.00	31.03	60.11	20.00	20.00	20.00	20.00	20.00
	M-15	4.02	0.50	2.00	2.00	2.00	2.00	2.00	60.02	16.12	20.00	20.00	20.00	20.00	20.00	4.77	1.45	2.00	2.00	2.00	2.00	2.00	2.00	60.07	60.03	20.00	20.00	20.00	20.00	20.00
	M-16	1.70	0.09	2.00	2.00	2.00	2.00	2.00	50.51	60.09	20.00	20.00	20.00	20.00	20.00	1.60	0.12	2.00	2.00	2.00	2.00	2.00	2.00	31.57	60.14	20.00	20.00	20.00	20.00	20.00
	M-17	2.25	0.62	2.00	2.00	2.00	2.00	2.00	49.03	60.03	20.00	20.00	20.00	20.00	20.00	2.80	0.61	2.00	2.00	2.00	2.00	2.00	2.00	38.15	60.08	20.00	20.00	20.00	20.00	20.00
	M-18	2.32	0.41	2.00	2.00	2.00	2.00	2.00	60.05	66.22	20.00	20.00	20.00	20.00	20.00	2.58	0.31	2.00	2.00	2.00	2.00	2.00	2.00	60.06	60.08	20.00	20.00	20.00	20.00	20.00
	M-19	1.83	0.41	2.00	2.00	2.00	2.00	2.00	6.44	60.03	20.00	20.00	20.00	20.00	20.00	2.06	1.06	2.00	2.00	2.00	2.00	2.00	2.00	6.06	60.03	20.00	20.00	20.00	20.00	20.00
	M-20	4.70	2.45	2.00	2.00	2.00	2.00	2.00	60.05	60.05	20.00	20.00	20.00	20.00	20.00	4.07	4.58	2.00	2.00	2.00	2.00	2.00	2.00	60.05	60.06	20.00	20.00	20.00	20.00	20.00
	M-21	0.49	0.05	2.00	2.00	2.00	2.00	2.00	9.92	50.83	20.00	20.00	20.00	20.00	20.00	0.60	0.01	2.00	2.00	2.00	2.00	2.00	2.00	9.44	60.09	20.00	20.00	20.00	20.00	20.00
	M-22	1.30	0.05	2.00	2.00	2.00	2.00	2.00	5.94	7.49	20.00	20.00	20.00	20.00	20.00	1.37	0.05	2.00	2.00	2.00	2.00	2.00	2.00	4.75	44.58	20.00	20.00	20.00	20.00	20.00
	M-23	2.11	0.20	2.00	2.00	2.00	2.00	2.00	60.03	-	20.00	20.00	20.00	20.00	20.00	2.56	0.53	2.00	2.00	2.00	2.00	2.00	2.00	57.29	60.08	20.00	20.00	20.00	20.00	20.00
	M-24	1.79	0.08	2.00	2.00	2.00	2.00	2.00	60.05	60.14	20.00	20.00	20.00	20.00	20.00	2.00	0.25	2.00	2.00	2.00	2.00	2.00	2.00	60.12	-	20.00	20.00	20.00	20.00	20.00
	M-25	0.79	0.03	2.00	2.00	2.00	2.00	2.00	10.96	32.08	20.00	20.00	20.00	20.00	20.00	1.08	0.06	2.00	2.00	2.00	2.00	2.00	2.00	11.08	60.09	20.00	20.00	20.00	20.00	20.00
	M-26	0.60	0.02	2.00	2.00	2.00	2.00	2.00	0.94	0.53	20.00	20.00	20.00	20.00	20.00	0.65	0.03	2.00	2.00	2.00	2.00	2.00	2.00	1.41	0.74	20.00	20.00	20.00	20.00	20.00
	M-27	1.26	0.03	2.00	2.00	2.00	2.00	2.00	4.20	60.03	20.00	20.00	20.00	20.00	20.00	1.01	0.05	2.00	2.00	2.00	2.00	2.00	2.00	5.03	60.06	20.00	20.00	20.00	20.00	20.00
	M-28	1.71	0.22	2.00	2.00	2.00	2.00	2.00	33.27	60.09	20.00	20.00	20.00	20.00	20.00	2.24	0.30	2.00	2.00	2.00	2.00	2.00	2.00	47.31	60.08	20.00	20.00	20.00	20.00	20.00
	M-29	3.56	0.94	2.00	2.00	2.00	2.00	2.00	60.07	60.05	20.00	20.00	20.00	20.00	20.00	4.68	12.31	2.00	2.00	2.00	2.00	2.00	2.00	60.09	-	20.00	20.00	20.00	20.00	20.00
	M-30	0.75	0.03	2.00	2.00	2.00	2.00	2.00	1.70	2.88	20.00	20.00	20.00	20.00	20.00	0.65	0.08	2.00	2.00	2.00	2.00	2.00	2.00	2.08	10.39	20.00	20.00	20.00	20.00	20.00

Continued on next page





Table B-3. Solving time of model performance comparison test (cont.)

Scale	Type Net. ID	1/∞							n/∞							1/n							n/n						
		CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5	CP	IP	DE-1	DE-2	DE-3	DE-4	DE-5
Large	L-01	0.93	0.05	2.00	2.00	2.00	2.00	2.00	43.92	60.16	20.00	20.00	20.00	20.00	20.00	1.02	0.05	2.00	2.00	2.00	2.00	2.00	32.16	-	20.00	20.00	20.00	20.00	20.00
	L-02	1.40	0.08	2.00	2.00	2.00	2.00	2.00	60.20	-	20.00	20.00	20.00	20.00	20.00	1.46	0.11	2.00	2.00	2.00	2.00	2.00	60.06	-	20.00	20.00	20.00	20.00	20.00
	L-03	0.93	0.03	2.00	2.00	2.00	2.00	2.00	60.10	60.06	20.00	20.00	20.00	20.00	20.00	1.03	0.03	2.00	2.00	2.00	2.00	2.00	60.05	-	20.00	20.00	20.00	20.00	20.00
	L-04	1.77	0.08	2.00	2.00	2.00	2.00	2.00	60.21	-	20.00	20.00	20.00	20.00	20.00	1.98	0.06	2.00	2.00	2.00	2.00	2.00	60.12	-	20.00	20.00	20.00	20.00	20.00
	L-05	0.47	0.03	2.00	2.00	2.00	2.00	2.00	58.95	-	20.00	20.00	20.00	20.00	20.00	0.66	0.03	2.00	2.00	2.00	2.00	2.00	49.63	-	20.00	20.00	20.00	20.00	20.00
	L-06	0.11	0.02	2.00	2.00	2.00	2.00	2.00	18.77	60.03	20.00	20.00	20.00	20.00	20.00	0.12	0.02	2.00	2.00	2.00	2.00	2.00	5.56	60.20	20.00	20.00	20.00	20.00	20.00
	L-07	0.51	0.05	2.00	2.00	2.00	2.00	2.00	7.71	-	20.00	20.00	20.00	20.00	20.00	0.72	0.05	2.00	2.00	2.00	2.00	2.00	11.99	60.11	20.00	20.00	20.00	20.00	20.00
	L-08	1.18	0.08	2.00	2.00	2.00	2.00	2.00	60.20	60.11	20.00	20.00	20.00	20.00	20.00	1.60	0.06	2.00	2.00	2.00	2.00	2.00	60.05	-	20.00	20.00	20.00	20.00	20.00
	L-09	1.32	0.08	2.00	2.00	2.00	2.00	2.00	60.11	60.14	20.00	20.00	20.00	20.00	20.00	1.51	0.12	2.00	2.00	2.00	2.00	2.00	60.11	-	20.00	20.00	20.00	20.00	20.00
	L-10	0.99	0.02	2.00	2.00	2.00	2.00	2.00	8.16	4.16	20.00	20.00	20.00	20.00	20.00	1.14	0.03	2.00	2.00	2.00	2.00	2.00	12.21	31.89	20.00	20.00	20.00	20.00	20.00
	L-11	1.07	0.05	2.00	2.00	2.00	2.00	2.00	50.43	60.11	20.00	20.00	20.00	20.00	20.00	1.20	0.06	2.00	2.00	2.00	2.00	2.00	44.14	60.11	20.00	20.00	20.00	20.00	20.00
	L-12	1.81	0.14	2.00	2.00	2.00	2.00	2.00	60.07	60.16	20.00	20.00	20.00	20.00	20.00	1.82	0.22	2.00	2.00	2.00	2.00	2.00	60.13	-	20.00	20.00	20.00	20.00	20.00
	L-13	2.69	0.27	2.00	2.00	2.00	2.00	2.00	60.08	60.14	20.00	20.00	20.00	20.00	20.00	3.09	0.42	2.00	2.00	2.00	2.00	2.00	60.19	-	20.00	20.00	20.00	20.00	20.00
	L-14	1.69	0.11	2.00	2.00	2.00	2.00	2.00	60.11	-	20.00	20.00	20.00	20.00	20.00	1.96	0.14	2.00	2.00	2.00	2.00	2.00	60.22	-	20.00	20.00	20.00	20.00	20.00
	L-15	0.60	0.02	2.00	2.00	2.00	2.00	2.00	10.52	39.16	20.00	20.00	20.00	20.00	20.00	0.67	0.03	2.00	2.00	2.00	2.00	2.00	9.82	60.14	20.00	20.00	20.00	20.00	20.00
	L-16	0.74	0.03	2.00	2.00	2.00	2.00	2.00	60.07	-	20.00	20.00	20.00	20.00	20.00	1.01	0.08	2.00	2.00	2.00	2.00	2.00	59.04	60.06	20.00	20.00	20.00	20.00	20.00
	L-17	2.35	0.28	2.00	2.00	2.00	2.00	2.00	29.66	60.08	20.00	20.00	20.00	20.00	20.00	2.22	0.61	2.00	2.00	2.00	2.00	2.00	47.21	60.14	20.00	20.00	20.00	20.00	20.00
	L-18	0.31	0.02	2.00	2.00	2.00	2.00	2.00	1.00	1.77	20.00	20.00	20.00	20.00	20.00	0.58	0.00	2.00	2.00	2.00	2.00	2.00	1.40	10.06	20.00	20.00	20.00	20.00	20.00
	L-19	1.49	0.05	2.00	2.00	2.00	2.00	2.00	49.53	60.12	20.00	20.00	20.00	20.00	20.00	1.91	0.11	2.00	2.00	2.00	2.00	2.00	59.48	-	20.00	20.00	20.00	20.00	20.00
	L-20	1.19	0.02	2.00	2.00	2.00	2.00	2.00	60.17	60.20	20.00	20.00	20.00	20.00	20.00	1.24	0.02	2.00	2.00	2.00	2.00	2.00	60.19	-	20.00	20.00	20.00	20.00	20.00
	L-21	1.78	0.06	2.00	2.00	2.00	2.00	2.00	60.16	-	20.00	20.00	20.00	20.00	20.00	1.63	0.27	2.00	2.00	2.00	2.00	2.00	60.13	-	20.00	20.00	20.00	20.00	20.00
	L-22	1.09	0.03	2.00	2.00	2.00	2.00	2.00	11.48	60.02	20.00	20.00	20.00	20.00	20.00	1.12	0.03	2.00	2.00	2.00	2.00	2.00	11.90	60.08	20.00	20.00	20.00	20.00	20.00
	L-23	0.96	0.02	2.00	2.00	2.00	2.00	2.00	60.03	60.16	20.00	20.00	20.00	20.00	20.00	0.91	0.03	2.00	2.00	2.00	2.00	2.00	60.18	-	20.00	20.00	20.00	20.00	20.00
	L-24	0.67	0.02	2.00	2.00	2.00	2.00	2.16	8.57	60.12	20.00	20.00	20.00	20.00	20.00	0.84	0.02	2.00	2.00	2.00	2.00	2.00	12.05	-	20.00	20.00	20.00	20.00	20.00
	L-25	0.47	0.02	2.00	2.00	2.00	2.00	2.00	2.15	-	20.00	20.00	20.00	20.00	20.00	0.56	0.02	2.00	2.00	2.00	2.00	2.00	2.33	38.09	20.00	20.00	20.00	20.00	20.00
	L-26	1.29	-	2.00	2.00	2.00	2.00	2.00	60.09	-	20.00	20.00	20.00	20.00	20.00	1.59	0.08	2.00	2.00	2.00	2.00	2.00	60.13	-	20.00	20.00	20.00	20.00	20.00
	L-27	1.36	0.06	2.00	2.00	2.00	2.00	2.00	60.15	60.12	20.00	20.00	20.00	20.00	20.00	1.55	0.08	2.00	2.00	2.00	2.00	2.00	60.06	60.09	20.00	20.00	20.00	20.00	20.00
	L-28	2.01	0.25	2.00	2.00	2.00	2.00	2.00	60.07	-	20.00	20.00	20.00	20.00	20.00	2.24	0.34	2.00	2.00	2.00	2.00	2.00	60.07	-	20.00	20.00	20.00	20.00	20.00
	L-29	1.09	0.03	2.00	2.00	2.00	2.00	2.00	42.17	60.09	20.00	20.00	20.00	20.00	20.00	1.22	0.05	2.00	2.00	2.00	2.00	2.00	26.61	60.03	20.00	20.00	20.00	20.00	20.00
	L-30	1.78	0.08	2.00	2.00	2.00	2.00	2.00	60.17	60.12	20.00	20.00	20.00	20.00	20.00	1.63	0.20	2.00	2.00	2.00	2.00	2.00	60.16	-	20.00	20.00	20.00	20.00	20.00