Abstract

This research deals with a set of robotic flow-shop scheduling problems with identical jobs, where robots are responsible for transferring jobs between machines. More particularly, we study the following three scheduling problems:

Problem 1: A problem where there is a single robot and job processing times are job-independent (but machine-dependent). The objective is to provide a machine and robot schedule that minimizes the makespan.

Problem 2: A combined robot selection and scheduling problem ($\mathcal{RSSP}$), where there are several robots moving on a single (shared) track. The robots are responsible for transporting identical jobs between identical machines. In the $\mathcal{RSSP}$, the objective of the scheduler is to (*i*) select robots from a given set of robot types; (*ii*) assign each selected robot to a subset of machines; and (*iii*) provide a schedule for each robot, in order to minimize two conflicting criteria: the makespan and the total robot selection cost. For a scheduling system with no-wait and no-idle restrictions, we analyze four different variations of the problem. The first is to find a schedule that minimizes the sum of the two conflicting criteria; the second and the third are to minimize one criterion, given an upper bound on the value of the other criterion; while the last variation consists of finding the entire set of Pareto-optimal points with respect to the two criteria.

Problem 3: A problem where there are two robots moving on parallel tracks and job processing times are job-independent (but machine-dependent). The objective is to provide a machine and robot schedule that minimizes the makespan.

Our main results for the three problems include:

- The construction of tight lower bounds for the makespan value of each of the three problems.
- For Problem 1 we construct a polynomial time procedure to minimize the makespan value, when there are three non-identical machines. The procedure is based on decomposing the problem to a set of sub-problems, and providing an optimal schedule for each sub-problem separately. Moreover, for the more general case, where the number of machines can be arbitrary, we construct a procedure which is based on collaborative reinforcement learning (RL) to obtain a near-optimal solution.
- For Problem 2 we prove that the first variation of the problem is solvable in polynomial time, while the other three variations are $\mathcal{NP}$-hard. For the $\mathcal{NP}$-hard variations, we show that a pseudo-polynomial time algorithm and a fully polynomial approximation scheme ($FPTAS$) exist, and derive three important special cases that are solvable in polynomial time.

- For Problem 3 we construct a heuristic procedure that is based on collaborative *RL* to obtain a near-optimal solution.

Our *main* contributions lies in the following two aspects.

- The definition and the theoretical analysis of the *RSSP*, which has never been analyzed before. In this problem we include some new features that have either rarely or have not been discussed at all in the literature. Among those features are several robot types, an arbitrary number of machines, a possibility to control the number of robots assigned to the production line and their assignment to machine sets, and a bicriteria approach for analysis. In fact, this is the first model that provides schedulers with a tool to optimally combine robot selection and scheduling decisions.

- The implementation of a new collaborative *RL* technique to solve robotic flow-shop scheduling problems, where robots are used as transportation resources in the shop. As far as we know, this technique has never been applied before to solve this set of problems. For the first problem with an arbitrary number of machines, we use a Robot-Adviser (*RA*) collaborative *RL* algorithm. In this case the robot can either operate autonomously (*AO*) or semi-autonomously (*SAO*), asking for advice from an adviser. The adviser is an agent that operates external to the system, and is simulated with different expertise levels. For the second problem we use a Robot-Robot (*RR*) collaborative *RL* algorithm, where the two robots are working together in one of the following collaboration modes: (*i*) *Full* , (*ii*) *Pull*, (*iii*) *Push*, or (*iv*) *None*. For both problems, the *RL* collaboration scheduling algorithms were programmed, and an extensive experimental study was done indicating that our collaborative *RL* algorithms can be used to determine near-optimal solutions with a small average gap from the lower bound value.

Key words: Scheduling, Flow-shop, Robotic flow-shop, Job transfer robots, Makespan, Reinforcement learning, Collaboration.

Contents

List of Figures

List of Tables

Nomenclature

A - The set of all possible actions.

$\bar{A}$ - A finite set of n elements $\{\bar{a}_1, \bar{a}_2, ..., \bar{a}_h\}$.

$\bar{A}_i$ - A subset of elements within $\bar{A}$.

a_i - A single action that belongs to set A.

$\breve{a}_u$ - The maximal cover of a single robot that is assigned to serve a set of machines that begins in machine M_u.

a^i - Sequences of robot moves in the order $M_1, ..., M_i$

$\widehat{a}^i$ - Sequences of robot moves in the order $M_i, M_{i+1}, ..., M_{m-1}$.

$\bar{a}_q$ - A single element that belongs to set $\bar{A}$.

b_q - The benefit from selecting element $\bar{a}_q$.

$\overline{C}$ - A given upper bound on the total cost.

$c_{(u,v)}$ - The minimum cost among all arcs that are directed from node u to v.

$c_{(u,v)_g}$ - The cost of arc $(u,v)_g$.

C_j- The completion time of job J_j.

C_{ave} - The learning performance measure.

$C_{\max}$- The makespan (the completion time of the last job at the last machine).

$\hat{C}_{\max}$ - The minimum average completion time achieved from the beginning of the learning trial until current learning episode.

C^m_{ij}- The completion time of the processing operation of J_j on M_i.

C^r_{ij} - The completion time of the transferring operation of J_j from M_i to M_{i+1}.

$C_{\max}$(adviser) - The makespan value of a solution provided by the adviser.

$C_{\max}(S)$ - The makespan value of a feasible solution S.

$C_{\max}(\acute{E}_i)$ - The makespan in learning episode i.

$C(IB_i)$ - The capacity of the input buffer to machine M_i.

$C(OB_i)$ - The capacity of the output buffer beside of machine M_i.

$C(P)$ - The total cost of a path P.

$\overline{D}$ - A given upper bound on the total duration.

D_i - An indicator for the success/failure of adviser's i'th advice.

d_j- The due date of job J_j.

$d_{(u,v)}$ - The minimum duration among all arcs that are directed from node u to v.

$d_{(u,v)_g}$ - The duration of arc $(u,v)_g$.

$\overline{d}_{(u,v)}$ - The maximum duration among all arcs that are directed from node u to v.

$D(P)$ - The total duration of a path P.

D_{UB} - An upper bound on the duration of any path in the graph $G(V,E)$.

e_i - An empty move of the robot from machine M_{i+1} to machine M_i.

E - The set of arcs within $G(V, E)$.

$\overline{E}$ - The set of arcs within $G(\overline{V}, \overline{E})$.

$E(u, v)$ - A set of arcs between two nodes u and v with $v > u$.

$\acute{E}$ - Learning session, which is a set of learning episodes.

$\acute{E}_i$ - Learning episode i from the current learning session.

$\breve{E}_j$- The earliness of job J_j.

$f_{\max}$- The maximal performance measure among all jobs.

F - The objective function.

F_j- The flow time of job J_j.

$f_j(C_j)$ - Job's J_j performance measure (which is a function of its completion time).

Fm- Flow-shop machine environment.

$Fm, R1$- Flow-shop machine environment with a single robot.

Fm, Rk - Flow-shop machine environment with k robots.

$f(s_t, a_t)$ - Function of state (s_t) and action (a_t) at time step t.

$f(v, D)$ - The minimum cost path with a duration not greater than D.

G_v - The length of the shortest path in a subgraph that includes the vertices $0, ..., v$.

$G_v(k)$ - The shortest duration path among all paths with exactly k arc in a subgraph that includes vertices $0, ..., v$.

$G(V, E)$ - A graph where V is the set of vertices and E is the set of arcs.

H_a - The scaled threshold of adviser, between $[0, 1]$.

H_{ai} - The weighted total advice over all advice instances up to the i'th instance.

IB_i - The input buffer of machine M_i.

$\mathcal{J}$ - A set of jobs.

J_j - A single job in set $\mathcal{J}$.

Jm - Job shop machine environment.

(J_j,M_i) - An operation where job J_j is transferred from M_i to M_{i+1}.

k - The number of robots.

$l_{(u,v)}$ - The minimum arc weight among all arcs from u to v.

L_j - The lateness of job J_j.

$L_{\max}$ - The maximal lateness.

LB_f - A lower bound that is based on the processing on M_f.

LB_r - A lower bound that is based on the robot movements.

Lr - Location of the robot.

$\mathcal{M}$ - A set of machines.

$\mathcal{M}_r$ - A set of machines designated to robot R_r ($\mathcal{M}_r \subseteq \mathcal{M}$).

M_i - A single machine that belongs to set $\mathcal{M}$.

$M_{l_{r+1}}$ - The last machine in set $\mathcal{M}_r$ that is also the first machine in $\mathcal{M}_{r+1}$.

n - The number of jobs to schedule.

O_j - The set of operations that belongs to job J_j.

O_{ij} - The operation of job J_j to be processed on machine M_i.

OB_i - The output buffer of machine M_i.

$O()$ - A notation of the approximated number of computational steps of an algorithm.

P - A path from vertex 1 to vertex N in $G(V, E)$.

P' - The control policy.

p_{ij} - The processing time of job J_j in machine M_i.

Pm - Identical parallel machines system.

$Prob_t(a_i)$ - The probability of selecting a feasible action a_i at time step t.

Q - The number of robot types.

Qm - Uniform parallel machines system.

$Q_t(a_i)$ - The reward estimation of selecting action a_i at time t.

$Q(s_t, a_t)$ - The reward estimation of a state-action pair.

r'_j - The ready time of job J_j.

$\{r\}$ - The type of robot R_r.

$r(s_t, a_t)$ - The reward when visiting state s_t and selecting an action a_t.

r_t - The numerical reward signal achieved at time t.

r_{ti} - The numerical reward signal achieved at time t when the robot serves machine M_i.

r_{tki} - The numerical reward signal achieved at time t when robot R_k serves machine M_i.

$\mathcal{R}$ - A set of robots.

R_k - A single robot that belongs to set $\mathcal{R}$.

RCm - Robotic cells environment with m machines.

Rm - Unrelated parallel machines system.

S - The set of all possible situations (states).

s_i - A single state that belongs to set S.

S_0 - The initial state of the system.

$S_{\check{T}}$ - The final state of the system.

S^m_{ij} - The start time of the processing operation of job J_j on machine M_i.

S^r_{ij} - The start time of transferring operation of job J_j from machine M_i to machine M_{i+1}.

Ss_x- The x subsequence of moves.

t^t - The system's transition time at time t.

$t_{i\{r\}}$- The transportation time required for robot R_r (designated on $\mathcal{M}_r$) of type $\{r\}$ to transfer a job from M_i to M_{i+1}.

$te_{i\{r\}}$- The time required for robot R_r (designated on $\mathcal{M}_r$) of type $\{r\}$ to move empty from M_{i+1} to M_i.

t_i - The transportation time (for a single robot type) required to transfer a job from M_i to M_{i+1}

te_i - The time required (for a single robot type) to move empty from M_{i+1} to M_i.

t_{iq} - The transportation time required for a robot of type q to transfer a job from M_i to M_{i+1}.

te_{iq} - The time required for a robot of type q to move empty from M_{i+1} to M_i.

t_{ijr} - The transportation time required for robot R_r to transfer job J_j from OB_i to IB_{i+1}

te_{ir} - The time required for robot R_r to move empty from machine M_{i+1} to machine M_i.

T - A set of all tardy jobs in a given schedule.

T_j - The tardiness of job J_j.

$T_{\max}$- The maximal tardiness.

$Temp$ - The temperature factor from Gibbs or Boltzmann distribution.

TRC - The total robot cost.

$TRC(S)$ - The total robots cost a of feasible solution S.

$T(n)$ - The running time of algorithm.

$T(S)$ - The variable part of the makespan value in a feasible solution S.

$(u, v)_g$ - A single arc from node u to v.

V - The set of vertices within $G(V, E)$.

$\hat{w}_j$- The weight of job J_j, indicating the job's priority.

w_q - The weight of element $\bar{a}_q$.

Z - A sequence of robot moves.

Z^* - The optimal sequence of robot moves.

z_λ- The λ'th transportation of the robot.

λ - The eligibility trace.

δ_q - The cost of a single robot of type q.

$\delta_{\{r\}}$ - The cost of robot R_r of type $\{r\}$, designated to machine set $\mathcal{M}_r$.

$\Delta(x)$ - An addition to the lower bound value.

$\Delta(x^*)$ - The minimum addition to the lower bound value.

τ - The time at which transfer is performed.

$\alpha|\beta|\gamma$ - Three-field problem notation.

α - The machine environment.

β - The processing characteristics and constraints.

γ - The optimizing criteria (objective function).

α' - The smoothing constant.

$\bar{\alpha}$ - The learning rate.

$\bar{\beta}$ - A positive number.

$\bar{\gamma}$ - The discount rate for future Q-values.

θ - A random number sampled from a normal distribution.

.

List of Abbreviations

ACO - Ant colony.

AO - Autonomously.

DV - Decision version

DP - Dynamic programming.

$FIFO$ - First in first out.

$FPTAS$ - Fully polynomial-time approximation scheme.

GA - Genetic algorithm.

IR - Identical robots.

IRT - Identical robots type.

IM - Identical machines.

LB - Lower bound.

MAS - Multi-agent system.

$MARL$ - Multi-agent reinforcement learning.

MC - Monte Carlo.

ML - Machine learning.

MRS - Multi-robot system.

NIM - Non-identical machines.

NIR - Non-identical robots.

$NIRT$ - Non-identical robots types.

$PTAS$ - Polynomial-time approximation scheme.

RA - Robot-adviser.

RL - Reinforcement learning.

RR - Robot-robot.

$RSSP$ - Robot selection and scheduling problem.

SA - Simulated annealing.

SAO - Semi- autonomously.

SPT - Shortest processing time.

SPP - Shortest path problem.

TD - Temporal-difference.

UB - Upper bound.

1 Introduction

"Scheduling is the allocation of resources over time to perform a collection of tasks" [48] . Scheduling is a decision-making function, it is the process of determining a schedule. In this sense, much of what was learned about scheduling can apply to other kinds of decision-making and therefore has general practical value. Scheduling becomes relevant in a situation where the characteristics of the tasks to be scheduled have been described and the configuration of the resources available has been determined. Scheduling theory is concerned primarily with mathematical models that relate to the scheduling function and the development of useful models and techniques, which has been the continuing interface between theory and practice. The theoretical perspective is particularly a quantitative approach, one that attempts to capture problem structure in concise mathematical form. In particular, this quantitative approach begins with a translation of decision-making goals into an explicit objective function and decision-making restrictions into explicit constraints [48].

Our research is devoted to study several robotic flow-shop scheduling problems. We aim to construct exact polynomial time algorithms, when it is possible. Otherwise, we use non-exact methods to obtain near-optimal solutions, by either applying methods that enable us to control the gap from the optimal solution (such as approximation schemes), or by applying collaborative reinforcement learning (RL) techniques. The introduction is thus divided into the following four subsections: (i) general introduction to the field of deterministic scheduling; (ii) an introduction to the field of computational complexity, which provides tools to estimate the difficulty of our problems and the efficiency of the suggested algorithms; (iii) an introduction to the field of RL; and (iv) an introduction to the field of multi-agent collaboration.

1.1 Deterministic scheduling problem

A deterministic scheduling problem is concerned with scheduling a set of n jobs $J \in \{J_1, J_2, ..., J_n\}$ on m machines $M \in \{M_1, M_2, ..., M_m\}$, where O_{ij} refers to the operation of job J_j on machine M_i for $j = 1, ..., n$ and $i = 1, ..., m$. It is usually assumed that each machine can process at most one job at a time and that each job can be processed on at most one machine at a time. The following data provides a typical input for various robotic flow-shop scheduling problems:

- **Release date (r'_j)** – The point in time at which job J_j becomes available for processing.

- **Processing time** (p_{ij}) – The time required for processing job J_j on machine M_i.

- **Due date** (d_j) – The point in time at which job J_j is due to be completed.
- **Weight** $(\hat{w}_j)$ – Predetermined factor that indicates the relative importance of job J_j.
- **Job Transport time** (t_{ijr}) – The transportation time required for robot R_r to transfer job J_j from OB_i to IB_{i+1}.
- **Robot empty return time** (te_{ir}) – The time required for robot R_r to move empty from machine M_{i+1} to machine M_i.

Throughout this research we will use the standard three field notation $\alpha|\beta|\gamma$, introduced by [33] for scheduling problems. The α field describes the machine environment and contains a single entry. The β field exhibits the processing characteristics and constraints and may contain no entry, a single entry, or multiple entries. The γ field contains the scheduler's objective.

1.1.1 Various scheduling systems (environments)

The α field describes the machine environment. Several possible machine environments are presented below:

- **Single machine** $(\alpha = 1)$ – Only one machine processes all the jobs. This is the simplest machine environment, which is a special case of all other machine environments.
- **Parallel machines -** $(\alpha = Pm, Qm, Rm)$ There are three types of parallel machine systems: identical $(\alpha = Pm)$, uniform $(\alpha = Qm)$, and unrelated $(\alpha = Rm)$, which differ by the definition of the processing time.
- **Flow-shop** $(\alpha = Fm)$ – There are m machines in a series. Each job has to be processed on each of the m machines in identical sequence. After completion on one machine, a job joins the queue of the next machine.
- **Flow-shop system with a single robot** $(\alpha = Fm, R1)$ **-** A flow-shop scheduling environment where loading, unloading, and the transportation of a job from one machine to the next is done by a single robot. In such a system all parts are available in front of machine M_1 and there is no need to transfer them to a storage station after unloading them from machine M_m.
- **Robotic cell** $(\alpha = RCm)$ **-** The definition of a robotic cell is identical to the definition of a flow-shop system with a robot with the exception that there exists an input station $(M_0 = IN)$ in front of the first machine where the raw material for the parts is available

in unlimited quantity and an output station ($M_{m+1} = OUT$) after the last machine where the finished parts can be stored in unlimited quantity.

- **Job-shop ($\alpha = Jm$) -** job-shop environment includes m machines in which each job has its own route to follow. It is usually assumed that each job visits each machine at most one time.

- **Flow-shop system with multi-robots ($\alpha = Fm, Rk$) -** A flow-shop scheduling environment consists of several robots, where each robot can load, unload, and transport a job from one machine to the next. Multi-robot systems are characterized by the (i) type of the robot (identical or non-identical); (ii) speed of movement; and (iii) type of movement (parallel/same track, etc.) In such a system all parts are available in front of machine M_1 and there is no need to transfer them to a storage station after unloading them from machine M_m.

1.1.2 Jobs characteristics and constraints

Each scheduling problem may have unique processing characteristics and constraints (included in the β field of the three field notation). Next, we present several such possible processing characteristics and constraints and their notation (in brackets).

- **Release dates (r_j')** - Each job may be specified with a different release date.

- **Permutation ($prmu$) -** Constraint related to a flow-shop environment, which indicates that queues in front of machine M_i operate according to *first in first out* (*FIFO*) rule.

- **Job independent processing times ($p_{ij} = p_i$)** – The job processing time is job-independent and machine-dependent.

- **Uniform processing time** ($p_{ij} = p$). This characteristic indicates that all jobs have identical processing time on all the machines.

Any other entry that appears in the β field is self-explanatory.

1.1.3 Performance measures and objectives functions (criteria)

Performance measures enable the scheduler to compare the quality of various possible schedules. The performance measure $f_j(C_j)$ of any job J_j is a function of the job's completion time, C_j, which is equal to the time at which the processing of job J_j is finished on the last machine. Examples of commonly used performance measures for any job J_j include: the job's completion time for which $f_j(C_j) = C_j$; the job's flow time for which $f_j(C_j) = F_j = C_j - r'_j$; the job's lateness for which $f_j(C_j) = L_j = C_j - d_j$, the job's tardiness for which $f_j(C_j) = T_j = \max\{0, L_j\}$, and the job's earliness for which $f_j(C_j) = \breve{E}_j = \max\{0, -L_j\}$.

Performance measures, like the ones described above, refer to a single job within a schedule. In order to evaluate the entire schedule, one must account for all the scheduled jobs. There are several ways to evaluate the quality of an entire schedule. Each way serves a different objective of the scheduling process, as reflected by a different objective function (criterion). In general, since each measure is a function of the job's completion time, the objective function can be referred to as $F = f(C_1, C_2, ..., C_n)$. There are two main *families* of objective functions. The first, $F = f_\Sigma = \sum_{j=1}^{n} f_j(C_j)$, is a summation over a specific measure value of all the jobs, while the second, $F = f_{\max} = \max\limits_{j=1,...,n} \{f_j(C_j)\}$, takes the maximal measure value among all jobs, where $f_j(C_j)$ can take the form of any performance measure. When using the three field notation we include the objective function F in the γ fields.

An important class of criteria is called the class of *regular* criteria and is defined below.

Definition 1 *An objective function F is* regular *if F is a non-decreasing function of C_j for $j = 1, ..., n$ (Pinedo [78]).*

Below are two examples of a regular scheduling criterion.

- **The Makespan** ($C_{\max} = \max\limits_{j=1,...,n} \{C_j\}$). The makespan is equivalent to the completion time of the last job on the last machine. A minimization of the makespan usually implies a high utilization of machines.

- **Total weighted flow time ($\sum_{j=1}^{n} \hat{w}_j F_j$) -** The total weighted flow time represents the total holding cost when each job has a different holding (flow) cost per unit of time, denoted by $\hat{w}_j$.

Below is an example of a non-regular criterion.

- **Total weighted earliness** ($\sum_{j=1}^{n} \hat{w}_j \breve{E}_j$). This objective function is characteristic of a scheduling problem in which each job has a penalty cost per unit of earliness time.

1.2 Computational complexity theory

Complexity theory is a discipline that deals with calculating and classifying the difficulty of a problem. It enables us to approximate the number of computational steps required to solve a specific problem, using a specific solution algorithm. The problems that are usually addressed are combinatorial optimization problems, for which scheduling problems is an important subclass.

In complexity theory, and in this text accordingly, the term *problem* refers to a generic description of a problem. An *instance* of a problem is a problem with explicit numerical data. The term *algorithm* refers to a proposed solution method to a given problem. The performance of any algorithm is commonly measured by two major aspects. The first is the quality of the solution it provides, and the second is how fast it is (as a function of the instance size).

Computational complexity theory deals with calculating the running time of a proposed algorithm. By calculating the running time of an algorithm, one could establish whether an algorithm is efficient in comparison to other known algorithms, or to a computer's capabilities. In this section, the theory of computational complexity is briefly introduced. First, we introduce the $O()$ notation, which is used in complexity theory to approximate the running time of an algorithm. Then, we discuss the common method to determine the complexity of a problem, present the definitions of complexity classes $\mathcal{P}$ and $\mathcal{NP}$, and present the formal definition of polynomial reduction. This is followed by presenting the definition of $\mathcal{NP}$-completeness, $\mathcal{NP}$-hardness, strong $\mathcal{NP}$-hardness, and their connection to pseudo-polynomial time algorithms.

1.2.1 Running time computation

As stated, a fundamental application of complexity theory is calculating the running time of an algorithm, as defined below:

Definition 2 *The running time of an algorithm is measured by the number of computational steps needed for obtaining a solution in the worst-case scenario.*

A computational step is an action performed by a computer that does not depend on the input size. Possible actions are simple mathematical operations, comparing two numbers, and so on. The term *worst-case* means that the algorithm could presumably run for much less on certain inputs, perhaps on most inputs. The computational complexity of an algorithm is usually approximated using the $O()$ notation as defined below:

Definition 3 *A function $f(n)$ is considered to be $O(g(n))$ if there are constants c and $\widetilde{n}$ such that $f(n) \leq cg(n)$ for all $n \geq \widetilde{n}$.*

The $O()$ notation is used to classify algorithms by how they respond in their running time to changes in the input size. When analyzing an algorithm using the $O()$ notation, the growth rate of the algorithm is the most important information. The growth rate determines how fast the number of needed computational steps will grow with respect to the size of the input. In order to determine the complexity of an algorithm, only the fastest growing element in the initial complexity calculation (denoted by $T(n)$) is considered. For example, for an algorithm with a running time of $T(n) = c_1n + c_2n^5$ it is clear that the dominant term is c_2n^5. Even if the constant c_1 is much greater than c_2, when the size of n increases, the element c_1n remains negligible and the factor c_2 can be omitted as well, regardless of its value. Hence, the algorithm's complexity is approximated by $O(n^5)$. Below, we provide an example of an order of computational time:

$$O(\log(n)) \prec O(n) \prec O(n^a) \prec O(n^b) \prec O(a^n) \prec O(b^n) \prec O(n!) \prec O(n^n), \text{ where } 0 < a < b.$$

When computing the complexity of a sum or a product of functions using the $O()$ notation, the following rules apply:

$$O(g(n)) + O(f(n)) = O(g(n) + f(n)) = O(\max\{g(n), f(n)\}),$$
$$O(g(n)) \times O(f(n)) = O(g(n) \times f(n)).$$

Next, we describe the most accepted way for classifying the complexity of an algorithm.

Definition 4 *An algorithm with the running time of $T(n) = O(g(n))$ is considered to be a polynomial-time algorithm if $g(n)$ is bounded by a polynomial function of n (at any power). Otherwise, it is considered to be an exponential time algorithm.*

A polynomial-time algorithm is preferable and considered efficient since a computer may not be able to perform an exponential-time algorithm in a reasonable length of time, especially when dealing with larger instances. Table 1 (taken from Garey, Johnson, and Sethi [27]) emphasizes the difference between polynomial and exponential time algorithms for different instance sizes (the table is based on a computer that can perform 10^6 computational steps per second).

1.2.2 Complexity classification

After the complexity of an algorithm has been defined, the complexity of a problem can be defined as follows:

Definition 5 *The complexity of a problem is equal to the complexity of the most efficient (in terms of complexity order) algorithm that solves the problem.*

Time complexity functions	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$
n	0.00001sec	0.00002sec	0.00003sec	0.00004sec	0.00005sec
n^2	0.0001sec	0.0004sec	0.0009 sec	0.0016 sec	0.0025 sec
n^3	0.001sec	0.008sec	0.027 sec	0.064 sec	0.125 sec
n^5	0.1 sec	3.2 sec	24.3 sec	1.7 min	5.2 min
2^n	0.001 sec	1 sec	17.9 min	12.7 days	35.7 years
3^n	0.059sec	58sec	6.5 years	3855 centuries	$2 * 10^8$ centuries

Table 1: A comparison between the computational requirements of several polynomial and exponential time algorithms.

Next, another distinction between different problems is provided, which is the difference between a decision problem and an optimization problem.

Definition 6 *A problem is called a decision problem if the solution to the problem is either* Yes *or* No.

Definition 7 *An optimization problem has a goal of either minimization or maximization of a certain objective function. Its objective is to find the best solution out of all feasible solutions with respect to the objective function.*

There is a strong connection between the two types of problems defined above, as every optimization problem can be converted into a decision problem. The conversion is usually done by adding a parameter K to the problem, and asking whether there is a feasible solution for which the evaluated objective function is not greater (or smaller) than K. For example, the optimization problem can be to find a schedule that minimizes $\sum_{j=1}^{n} \hat{w}_j F_j$, while the converted decision problem would ask if there exists a schedule in which $\sum_{j=1}^{n} \hat{w}_j F_j \leq K$.

Complexity theory divides decision problems into classes according to their complexity as follows:

Definition 8 *The class $\mathcal{P}$ contains all the decision problems for which there exists an algorithm that leads to a correct yes-no answer in a polynomial time with respect to the size of the instance (n).*

Definition 9 *The class $\mathcal{NP}$ contains all the decision problems for which, given an appropriate "clue", the correct answer can be verified in a polynomial time with respect to the size of the instance (n).*

Clearly, $\mathcal{P} \subseteq \mathcal{NP}$, since for any problem in $\mathcal{P}$ we can simply ignore the clue and just solve the problem in polynomial time. Another important part of the $\mathcal{NP}$-hardness theory is the polynomial reducibility:

Definition 10 *Problem P reduces to problem Q if problem P can be solved by using an algorithm for the solution of problem Q as a subroutine. A polynomial-time reduction is a reduction that uses polynomial time excluding the time within the subroutine.*

A reduction from P to Q is denoted by $P \propto Q$ and may be used to show that Q is at least as difficult as P since whenever an efficient algorithm exists for Q, one exists for the P as well. Polynomial-time reductions are frequently used for defining and classifying problems as described in Definition 11 below.

Definition 11 *A decision problem P is said to be $\mathcal{NP}$-complete if P is in the $\mathcal{NP}$-class and all the problems in the $\mathcal{NP}$-class are reducible to P.*

It implies from Definition 19 that proving the $\mathcal{NP}$-completeness of a problem P can be done by showing that P belongs to the $\mathcal{NP}$ class and then reducing one of the known $\mathcal{NP}$-complete problems to problem P. It also implies that if two decision problems, P and Q, are both $\mathcal{NP}$-complete, then, $P \propto Q$ and $Q \propto P$. Hence, we can conclude from Definition 19 that either all $\mathcal{NP}$-complete problems are solvable in polynomial time (meaning $\mathcal{P} = \mathcal{NP}$), or none of them. The question whether $\mathcal{P} = \mathcal{NP}$ or not is still open, but today it is widely assumed that $\mathcal{NP}$-complete problems cannot be solved in polynomial time (i.e., $\mathcal{P} \neq \mathcal{NP}$).

There is an important connection between the complexity class of an optimization problem and the corresponding decision version of the problem, as shown by the following lemma.

Lemma 1 *The complexity of an optimization problem is determined according to the complexity of the corresponding decision problem. Hence, if we can reduce the decision version of an optimization problem Q to P and Q is $\mathcal{NP}$-complete, we say that P is $\mathcal{NP}$-hard.*

We can conclude from Definition 11 and Lemma 1 that the class of $\mathcal{NP}$-complete problems forms a subclass of the class of $\mathcal{NP}$-hard problems. Unlike $\mathcal{NP}$-complete, $\mathcal{NP}$-hard problems may be of any type including optimization problems. If an optimization problem can be solved in polynomial time, its related decision problem can be solved in polynomial time as well. We simply compare the value obtained from the solution of the optimization problem with the bound provided as input to the decision problem, meaning, if we can provide evidence that a decision problem is hard, we also provide evidence that its related optimization problem is hard.

The input of a problem varies between different types of problems. It comprises data and parameters needed to describe a specific instance of a problem. The complexity of a problem is defined with respect to the problem's input. Two possible ways to encode an input of a problem are binary and unary codes. In binary code each number of the input data is represented using the digits 0 and 1. In unary code the numbers are represented using only "1" digits. For example, the number 7 is represented by 3 digits in binary code and by 7 digits in unary code. A problem's complexity may vary according to its input encoding. Thus, it is possible that a problem is $\mathcal{NP}$-hard under the binary encoding scheme, but solvable in polynomial time under the unary encoding scheme. This difference leads to a distinction between $\mathcal{NP}$-hard problems according to the "strength" of their $\mathcal{NP}$-hardness, as explained below.

Definition 12 *A problem that is $\mathcal{NP}$-hard with respect to the binary encoding scheme but* not *the unary encoding scheme is said to be $\mathcal{NP}$-hard in the* ordinary *sense.*

Definition 13 *A problem is said to be $\mathcal{NP}$-hard in the* strong *sense if it is $\mathcal{NP}$-hard with respect to both the binary and the unary encoding schemes.*

That leads to the definition of pseudo-polynomial time algorithms as described below.

Definition 14 *A pseudo-polynomial time algorithm is an algorithm that runs in polynomial time with respect to the range of the possible input values and the number of elements in the input.*

It implies from Definition 14 that a pseudo-polynomial time algorithm runs in polynomial time when the input is encoded unary. The connection between the existence of a pseudo-polynomial time algorithm for solving a problem and the classification of the problem as $\mathcal{NP}$-hard in the strong sense is given in Definition 15 below:

Definition 15 *If an optimization problem Q is $\mathcal{NP}$-hard in the strong sense, then there cannot be any pseudo-polynomial time algorithms for Q (unless $\mathcal{P} = \mathcal{NP}$).*

It implies from Definitions 12 and 15 that if there is a pseudo-polynomial time algorithm for solving an $\mathcal{NP}$-hard problem P, then P is $\mathcal{NP}$-hard in the ordinary sense.

In real-life problems, even when a problem was proved to be an $\mathcal{NP}$-hard problem, in the ordinary or strong sense, a solution is needed. When the actual input size is small, an algorithm with exponential running time may be perfectly satisfactory. Even if searching all feasible solutions, scope should be considered. However, when the optimal solution cannot be achieved in a reasonable length of time, an algorithm that generates a relatively good solution is needed. The next subsection describes these types of algorithms, also known as heuristic algorithms, for which approximation algorithms are an important subclass.

1.2.3 Heuristic algorithms

An acceptable approach to provide a solution to $\mathcal{NP}$-hard optimization problems is by using heuristic algorithms.

Definition 16 *A heuristic algorithm is an algorithm that runs in a polynomial time. It provides a feasible solution but it does not necessarily provide the optimal solution.*

Definition 17 *An acceptable (feasible) solution to a problem, greater than or equal to the optimal solution, is defined as upper bound (UB) solution to the optimal solution.*

The quality of a heuristic algorithm is derived from the algorithm's running time and the quality of the suggested solution. The quality of a suggested solution is measured by its difference from the optimal solution. Thus, an important subclass of heuristic algorithms is approximation algorithms.

Definition 18 *Let A be a heuristic algorithm for a minimization problem $\mathcal{P}$ with a solution value of $F^A(\mathcal{I})$ for instance $\mathcal{I}$, and let $F^*(\mathcal{I})$ be the optimal (best) solution value for problem $\mathcal{P}$ with instance $\mathcal{I}$. Then, Algorithm A is a $\rho-$approximation if for any instance $\mathcal{I}$ of the problem we have that*

$$\frac{F^A(\mathcal{I})}{F^*(\mathcal{I})} \leq \rho. \tag{1}$$

Definition 18 indicates that the uniqueness of an approximation algorithm is that it guarantees that the solution value it generates is at most ρ times the value of the optimal solution. There are approximation algorithms that can achieve increasingly smaller ratio bounds by using more and more computation time. In those algorithms, there is a trade-off between computation time and the quality of the approximation. Two common and important types of such approximation algorithms are *polynomial-time approximation scheme* (**PTAS**), and *fully polynomial-time approximation scheme* (**FPTAS**), as defined below.

Definition 19 *Given $\varepsilon > 0$, a heuristic algorithm A for a minimization problem $\mathcal{P}$ is said to be a* polynomial-time approximation scheme *(or PTAS in short) if (i) the running time of the approximation is polynomial for n but not necessarily for $1/\varepsilon$, and (ii) for any instance $\mathcal{I}$ of $\mathcal{P}$, the condition in eq. (1) holds with $\rho = 1 + \varepsilon$.*

PTAS algorithms may not be practical for every ε value. Although the running time of the PTAS algorithm is polynomial for each fixed n, the exponent of the polynomial might depend strongly on ε. For $\varepsilon \rightarrow 0$, it is possible that the running time of the algorithm would become exponential. A more efficient approximation scheme is the FPTAS.

Definition 20 *Given $\varepsilon > 0$, a heuristic algorithm A for a minimization problem $\mathcal{P}$ is said to be a* fully polynomial-time approximation scheme *(or FPTAS in short) if (i) the running time of the approximation is polynomial for both n and $1/\varepsilon$, and (ii) for any instance $\mathcal{I}$ of $\mathcal{P}$, the condition in eq. (1) holds with $\rho = 1 + \varepsilon$.*

1.3 Reinforcement learning

In some parts of our research, when we could not provide an efficient procedure to solve the corresponding scheduling problem, we construct a procedure that is based on *RL* to obtain a near optimal solution. This section is devoted to introducing the method of *RL*, its importance, and to presenting some related applications for which this method has been successfully applied before.

Several machine learning (*ML*) methods, ranging from inductive logic programming to *RL*, have been applied to many subproblems in robot perception and control [25]. *RL* is about learning how to map situations ($s_i \in S$) to actions ($a_i \in A$) so as to maximize a numerical reward (r) signal ([87]). Here S represents the set of all possible situations (states), and A represents the set of all possible actions. In the process of learning, the agent (an agent is something, like a robot, that perceives and acts) interacts with an environment or a system. The agent is not told which action to take, but by trying several actions the agent must discover (learn) the actions that will yield the largest reward. This implies that the agent must have the abilities to sense the state of the environment and to take actions that affect the state. Furthermore, the agent must also have goals relating to the state of the environment. The two most important distinguishing features of *RL* are ([87]): (i) *trial and error search* - selecting action $a_i \in A$ has an effect on the immediate reward; and (ii) *delayed reward* - selecting action $a_i \in A$ may affect not only the immediate reward, but also the next situation and, through that, all subsequent rewards. The basic assumption in *RL* is that the system state, s_{t+1}, at step (time) $t + 1$, must be a function of only its last state, s_t, and the selected action, a_t; that is, $s_{t+1} = f(s_t, a_t)$ where $s_t \in S$ and $a_t \in A$ are the state and chosen action at time step t, respectively [79].

An *RL* system includes four main sub-elements ([83]): (i) *policy* - agent's way of behaving, a way of mapping situations to actions. The policy is the core of an *RL* agent since that alone is sufficient to determine behavior. (ii) *Reward function* - defines the goal in an *RL* problem and indicates what is good in an immediate sense. It maps each perceived state (or state-action pair) of the environment to a single number, a reward, indicating the intrinsic desirability of that state. (iii) *Value function* - value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state, indicating what is good in the long run. (iv) *Model of the environment* - the model mimics the behavior of the environment. The model predicts, given a state and action, the resultant next state and next reward. Models are used for planning, deciding on a course of action by considering possible future situations before they are actually experienced.

A challenging issue that exists in RL is the trade-off between exploration and exploitation [87]. In order to maximize the reward, an RL agent must choose between actions in set A, which were tried in previous activities, so as to produce the highest reward. Those actions in A may now be known to the agent (exploitation), but before that they would have to be discovered, meaning they were new actions (exploration). In other words, the agent must exploit what he already knows but he also has to explore the environment in order to make better action selections in the future, which may yield a greater reward. One way to achieve both exploration and exploitation is by using a stochastic method called ε-greedy action selection ([87]), which will be defined later.

1.3.1 Action selection

Part of the learning process is to decide whether to explore for new actions or make use of the experience already gained. Choosing exploring or exploiting in an unequivocal way will result in losing the optimal solution or probably converge to sub-optimal solutions. In order to provide an action selection rule, an RL process creates a probability distribution over the set of possible actions. This is done by assigning a numerical value (an estimation), $Q_t(a_i)$, to each possible action $a_i \in A$ at time t. Given a set of estimations at time t, $Q_t(a_i)$ for any $a_i \in A$, we can select one of many methods presented in the literature to select the next action, a_t. Below, we survey some of the most commonly used methods for action selection.

Greedy action selection This method always exploits the gained experience, i.e., it always selects the action with the highest estimated payoff. Thus, at time step t it chooses the action a_i for which $Q_t(a_i) = \max_{a_j \in A} \{Q_t(a_j)\}$. A significant disadvantage of this method is that it spends no time for exploring actions that seem inferior, and because of that the agent might get stuck at a sub-optimal results.

ε-greedy action selection (or near greedy) In this method, the agent, at any given time step t, selects action a_i for which $Q_t(a_i) = \max_{a_j \in A} \{Q_t(a_j)\}$ with a probability of $1 - \varepsilon$. Otherwise, with a probability of ε, the agent selects a random action from the set of all possible actions, where each action in this set has the same probability to be chosen. The disadvantage is that once a random action is selected, no exploitation is made, i.e., all actions have the same probability to be selected.

Softmax action selection In this method the greedy action is still given the highest probability of getting selected, but the other actions are ranked and weighted according to their estimates. The most common way to implement the softmax action selection method is by using the *Gibbs or Boltzmann distribution*, for which the probability, $Prob_t(a_i)$, of selecting a feasible action a_i at time step t is given by

$$Prob_t(a_i) = \frac{e^{Q_t(a_i)/Temp}}{\sum_{a_j \in A'} e^{Q_t(a_j)/Temp}} \tag{2}$$

where $Temp$ is a temperature factor (in fact in every iteration $Temp$ will be multiplied with a decay factor) that determines the amount of exploration, and A' includes all possible actions at time step t. High $Temp$ values will result in a lot of exploration, and low values will result in the exploitation of experience. In the limit, when $Temp \rightarrow 0$, softmax selection becomes the same as the greedy action selection.

1.3.2 *RL* algorithms

There are several different *RL* algorithms, each of which updates the set of estimators by applying a different approach. After updating the estimators each method may use any action selection method or predefined policy to move from one state to another. Next we briefly present some of the most commonly used algorithms.

The temporal-difference (TD) algorithm "If one had to identify one idea as central and novel to *RL*, it would undoubtedly be *temporal-difference* (*TD*) learning ([87])." *TD* learning [87] is a combination of *Monte Carlo* (*MC*) and *Dynamic Programming* (*DP*) ideas. *MC* methods require only experience-sample sequences of states, actions, and rewards from on-line or simulated interaction with an environment. In *DP*, the key idea is the use of value functions to organize and structure the search for good policies. *TD* methods, like *MC*, can learn directly from raw experience without a model of the environment's dynamics, and like *DP*, *TD* methods update estimates (V-values of estimates that follow predefined policy) based in part on other learned estimates, without waiting for a final outcome. The advantage of *TD*, compared to *DP*, methods is that they do not require a model of the environment, of its reward, and next-state probability distributions. When compared to *MC* methods their advantage is that they are naturally implemented on-line (one time step), in a fully incremental fashion. The relationship between *TD*, *DP*, and *MC* methods is a recurring theme in the theory of reinforcement learning; their ideas and methods blend into each other and can be combined in many ways.

The TD(λ) algorithm *TD*(λ) was developed by Richard Sutton [86]. In *TD*(λ), λ refers to the use of an eligibility trace, which is one of the basic mechanism of *RL*. According to [87] there are two ways to view λ: theoretically (forward view) and mechanistically (backward view). The approach of the theoretical view was presented as "For each state visited, we look forward in time to all the future rewards and decide how best to combine them." The mechanistic view "provides a causal, incremental mechanism for approximating the forward view and, in the off-line case, for achieving it exactly."

"*TD*(λ) is an elegant algorithm for approximating the expected long-term future cost (estimator) of a dynamic stochastic system as a function of the current state" ([9]). They develop a variation of *TD*(λ) to train an agent playing a two-player game, such as chess or backgammon, when *TD*(λ) "provides a way of using the scalar rewards such that existing supervised training techniques can be used to tune the function approximator" ([9]).

The SARSA & SARSA(λ) algorithm The name *SARSA* reflects the fact that the main function for updating the estimator (Q-value) at time t depends on: the current state, s_t; the action the agent chooses, a_t; the reward the agent gets for choosing this action, r_{t+1}; the state, s_{t+1}, that the agent will be in after taking action a_t; and the next action, a_{t+1}, that the agent will choose in its new state, s_{t+1}. Taking every letter for each parameter in the process $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ yields the word *SARSA*. *SARSA*, *TD* method, and *SARSA*(λ) are on-policy algorithms, meaning that they approximate the Q-value, the action values for the current policy, then improve the policy gradually based on the approximate values for the current policy. The policy improvement can be done in many different ways. The simplest approach is to use the ε-greedy policy with respect to the current action value estimates. Following the description above, the two algorithms will now be described separately and generally. *SARSA* algorithm [87] considers transitions from state-action pair to state-action pair, and learns the value of state-action pairs. *SARSA*(λ) algorithm is the eligibility trace version of *SARSA* ([87]). The idea in this method is to apply the *TD*(λ) prediction method to state-action pairs rather than to states, with use of a trace for each state-action pair.

The *Q*-learning algorithm "Q-learning, one of the most important *RL* algorithms, was presented by Watkins in 1989 ([89]). Some researchers classify the Q-learning algorithm as a special case of the *TD* learning algorithm" ([34]). The Q-learning algorithm works by estimating the values of state-action pairs, $Q(s_t, a_t)$, i.e., in this method the system directly estimates the Q-values and then uses them to derive a control policy using the local greedy strategy [87].

The algorithm: the first step is to initialize the system's action-value function, Q. Since no prior knowledge is available, the initial values can be arbitrary (e.g., uniformly zero). The second step is to initialize the control policy, *P'*; this is achieved by assigning to *P'* the action that locally maximizes the action-value. After performing the initialization, steps one and two, with every action (until stopping criterion is met) the agent is updating the action values $Q(s_t, a_t)$. For example: at time step t the agent visits state $s_t \in S$ and selects an action $a_t \in A$, receives from the process the reward $r(s_t, a_t)$ and observes the next state s_{t+1} and then updates the action value $Q(s_t, a_t)$.

The advantage is that the update rule is model free as it is a rule that just relates Q values to other Q values. It does not require a prior mapping from actions to states and it can calculate the Q values directly from the elementary rewards observed. The disadvantage is that the agent learns quite slowly because only one time-step is traced for each action.

$Q(\lambda)$-learning Q-learning only considers the immediate reward. It propagates the reward backward only one step. Unlike Q-learning, $Q(\lambda)$-learning not only considers the immediate reward, it also takes the discounted future rewards into consideration ([10]). Actually, a generalization of Q-learning, represented by $Q(\lambda)$ [74], uses eligibility traces, $e(s_t, a_t)$: the one-step Q-learning is a particular case with $\lambda = 0$ [31], where λ represents the eligibility decay rate. The greater λ is, the longer the sequence of values of state-action pairs updated. To boost learning, a multi-step tracing mechanism, the eligibility trace, is used in which the values of a sequence of actions can be updated simultaneously according to the respective lengths of the eligibility traces [96]. The advantage of the $Q(\lambda)$ algorithm is that the learning process is faster. The disadvantage is that the convergence of $Q(\lambda)$ is not assured for $\lambda > 0$ [31]. Furthermore, Sutton and Barto [87] indicate that unlike *TD*(λ) or *SARSA*(λ), Watkins's [89] $Q(\lambda)$ does not look ahead all the way to the end of the episode in its backup. It only looks ahead as far as the next exploratory action. Aside from this difference, however, Watkins's [89] $Q(\lambda)$ is much like *TD*(λ) and *SARSA*(λ). Their look ahead stops at episode's end, whereas $Q(\lambda)$'s look ahead stops at the first exploratory action, or at episode's end if there are no exploratory actions before.

1.3.3 Applications of *RL* algorithms

Table 2 below presents a few applications for which *RL* is applied to robot learning and scheduling problems.

Algorithm	**Application**	**Reference**
Q and $Q(\lambda)$-learning	Robot Navigation	[10], [96]
$TD(\lambda)$	Manufacturing scheduling	[94], [95]
Q-learning	Flow-shop Scheduling	[76]
Q-learning	Job-shop Scheduling	[8]
Q-learning	Multi-Robot Systems	[23], [6]
Q-learning	Collaboration and Scheduling	[29], [30]
$Q(\lambda)$-learning	Human-Robot Collaboration	[41]
Q-learning	Multi-Robot Collaborative Task	[75]

Table 2: Examples of applications of RL algorithms in robot learning and scheduling.

1.4 Multi-agent collaboration

In this thesis we use RL algorithms to solve various robotic scheduling problems. Our RL algorithms are unique in the sense that they exploit collaboration between different agents in the system (which might either be the robots themselves and/or an external adviser). In this subsection we provide a brief introduction to collaboration within multi-agent systems (MAS).

"Cooperative MAS are ones in which several agents attempt, through their interaction, to jointly solve tasks or to maximize utility. Due to the interactions among the agents, multi-agent problem complexity can rise rapidly with the number of agents or their behavioral sophistication. The challenge this presents to the task of programming solutions to MAS problems has spawned increasing interest in machine learning techniques to automate the search and optimization process ([65]). "The multi-agent learning is a field that refers to many areas, including RL, evolutionary computation, game theory, complex systems, agent modeling, and robotics" ([65]). Further, it has two features that differentiate them from ordinary ML: (i) due to interaction between agents small changes in learned behaviors can often result in unpredictable changes in the resulting multi-agent group as a whole; and (ii) there could be multiple learners, each learning and adapting in the context of others. Multi-agent learning can be divided into two categories ([65]): (i) *team learning* - applying a single learner to discover joint solutions to multi-agent problems; or (ii) *concurrent learning* - using multiple simultaneous learners, often one per agent, that attempt to improve parts of the team. In addition, there are two types of communication that are connected to learning: (i) *direct communication* - an external communication method that agents may share information with one another, such as: shared blackboards, signaling, and message-passing and; (ii) *indirect communication* - implicit transfer of information from agent to agent through modification of the world environment, such as leaving footsteps in snow.

Claus and Boutilier [12] distinguish between two cases of agent's actions: (i) *joint action* *learners* - agents that get information about their own choice of action as well as their partners' choices; and (*ii*) *independent learners* - agents that only know their own actions. The joint action learners suffer from combinatorial explosion of the size of the state-action space with the number of agents, while independent learners have three major difficulties ([51]): (*i*) other learning agents are unpredictable elements of the environment; (*ii*) the environment is no longer stationary; and (*iii*) it is hard to make sure that all independent learners coherently choose their individual actions such that the resulting joint action is optimal (i.e., how to coordinate between the agents).

Multi-robot systems (*MRS*) can also be defined as an *MAS*, where a group of autonomous, interacting entities (as robots) share a common environment, which they perceive with sensors and upon which they act with actuators (see, e.g., Weiss [35], and Thomas and Martin [26]). *MRS* have many advantages ([23]), such as: (*i*) they can often be used to fulfill the tasks that are difficult to be accomplished by an individual robot; and (*ii*) their performance with cooperation contributes to task solutions with a more reliable, faster, or cheaper way. Still, *MRS* cannot predict the entire potential situation they may encounter, and specify robots' optimal behaviors in advance. Therefore, robots in *MRS* must learn from, and adapt to their operating environment and their counterparts. *RL* is one of the *ML* methods that enable learning in *MRS*.

"*MAS* are rapidly finding applications in a variety of domains, including robotics, distributed control, telecommunications, and economics. The complexity of many tasks arising in these domains makes them difficult to solve with preprogrammed agent behaviors. The agents must, instead, discover a solution on their own, using learning. A significant part of the research on multi-agent learning concerns *RL* techniques" ([67]). In *MAS RL*, agents can compete or cooperate to accomplish the goal [20]. "In resource management, the robots in a collaborative team were one of the following: (*i*) each agent manages one resource - learns how to best service requests; and (*ii*) agents learn how to best select resources, when both is through optimization of a given performance measure. The performance measures can be minimum waiting time for resources, resource usage and etc." ([67]).

2 Literature Review

This section includes a literature review of the three main thesis topics: (i) flow-shop scheduling problems with transportation times (which is sometimes referred to as robotic flow-shop scheduling problems); (ii) implementation of RL algorithms to solve scheduling problems; and (iii) the uses of collaboration in RL. Based on gaps we identified in the review, we conclude with a subsection that presents our research directions.

2.1 Flow-Shop scheduling problems with transportation times

In most studies related to flow-shop scheduling, the common assumption is that job transfer times between any two consecutive stations (machines) is either not relevant or negligible. In such systems, once a job has finished its processing on machine M_i it is immediately available for processing on machine M_{i+1} for $i = 1, ..., m-1$. However, in many cases a robot is required for transfer operations. If the number of robots in the system is limited, the scheduler has to schedule the robot moves in addition to making the regular job scheduling decisions. As far as we know, the literature related to robot scheduling in flow-shop systems started in 1976 (see Phillips and Unger [77]) and since this seminal work many other works have been done on this subject.

In this section we give a general definition of flow-shop scheduling problems with transportation times and present different variants of this problem that have been studied in the literature. A flow-shop scheduling problem with transportation times can be defined as follows: A set of n independent jobs, $\mathcal{J} = \{J_1, ..., J_n\}$, is available for processing at time zero. The jobs are to be processed on a set of m machines, $\mathcal{M} = \{M_1, ..., M_m\}$, in a flow-shop scheduling system. In such a system, each job J_j consists of m operations $O_j = \{O_{1j}, ..., O_{mj}\}$, which must be processed in the order $O_{1j} \rightarrow O_{2j} \rightarrow \ldots \rightarrow O_{mj}$ on the m machines. The operation O_{ij} must be processed on machine M_i for $p_{ij} \geq 0$ time units and each machine can only process one job at a time. In front of each machine M_i $(i = 1, ..., m)$, there is an input buffer IB_i with a capacity of $C(IB_i)$, which implies that no more than $C(IB_i)$ jobs can be in this buffer at any given time. In order to process a job on M_i, the job has to be taken from the input buffer, and be uploaded on M_i. After the completion of the job on M_i, it has to be downloaded from the machine, into the output buffer OB_i beside M_i. The capacity of OB_i is denoted by $C(OB_i)$. It is usually assumed that there is an automatic mechanism beside each machine M_i $(i = 1, ..., m)$ that allows to performing both download and upload operations in negligible time. A set of k robots, $\mathcal{R} = \{R_1, ..., R_k\}$, provides transportation service between the machine buffers. Let t_{ijr} represent the transportation time required for robot R_r to transfer job J_j from OB_i to IB_{i+1}, and let te_{ir} be the time required for robot R_r to move empty without carrying a job from machine M_{i+1} to machine M_i. The empty return times are assumed to be additive, i.e., the time for the robot to travel between two distinct machines is the sum of the empty traveling times between all intermediate machines. The typical objective in this set of problems is to find a job and robot schedule to minimize (or maximize) a given scheduling criterion.

Robotic flow-shop systems are very complicated to analyze and thus in order to provide an exact (mathematical-based) analysis researchers used various simplified assumptions. The most common assumption is that there is a single robot that serves the entire production line (see, e.g., Stern and Vinter [84], Panwalkar [73], Levner *et al.* [60], Kise *et al.* [50], Agnetis [4], Hurink and Knust [38], and Ling and Guang [64]). Among the other commonly used assumptions is the case where the number of machines is limited to two (see, e.g., Yu [92], Dell'Amico [22], Karuno and Nagamochi [42], and Agnetis [4], Hurink and Knust [38]), that empty return times equal zero (see, e.g., Kise [49], Hurink and Knust [38], and Ling and Guang [64]), that loading and unloading times are zero (see, e.g., Kise [49], Agnetis [4], and Agnetis and Pacciarelli [5]), that job processing times are job-independent (see, e.g., Levner *et al.* [61], Kats *et al.* [45], Che *et al.* [13] and [14], and Chu [21]), that the production is cyclic (see, e.g., Levner *et al.* [61], Kats *et al.* [45], Agnetis [4], Che *et al.* [13] and [14], and Chu [21]), and that there is a sufficient number of robots with no technological constraints (see, e.g., Yu [92], Dell'Amico [22], and Karuno and Nagamochi [42]).

A good example for the wide range of assumptions used is the paper by Hurink and Knust [38], where it is assumed that only a single robot serves the entire set of machines and that there is an unlimited buffer between any two consecutive machines. Moreover, in several cases they even consider more restricted models of two machines, equal transportation times, zero empty return times, and equal and even unit processing times. We note that the robotic flow-shop scheduling problem is so complicated that even under these very restrictive assumptions, Hurink and Knust [38] showed that the problem remains strongly $\mathcal{NP}$-hard in most cases and exact (polynomial time) algorithms have been presented only for some special cases where *all* processing times are *equal*.

Next we review the related literature starting with works that consider the case of a single robot and moving to those that consider multiple robots.

2.1.1 Problems with a single transferring robot

There are several different variants of flow-shop scheduling problems where transportations are performed by a single robot. A literature review of the main variants appears below.

- *Variant 1*: The variant where there is an unlimited buffer between the machines, a single robot serves the entire set of m machines, and the objective is to minimize the makespan. Kise [49] proved that the problem with two-machines, a single robot, and equal transportation times is ordinary $\mathcal{NP}$-hard. Hurink and Knust [38] showed that the problem with two machines, a single robot, and zero empty return times is equivalent to the well-known $F3\,||\,C_{\max}$ problem, and is thus strongly $\mathcal{NP}$-hard. They also showed that the problem remains strongly $\mathcal{NP}$-hard even if either the transportation or the processing times are all equal. Ling and Guang [64] showed that the problem is also strongly $\mathcal{NP}$-hard if there are only two possible transportation times and the processing times are job-dependent and machine-independent. In contrast to these hardness results, Hurink and Knust [38] showed that other special cases of the problem can be solved in polynomial time. This includes the case where either *all* processing times are restricted to unity and the case of equal processing times with only two possible transportation times. Hurink and Knust further showed that if all processing times are equal and the transportation times are job-independent (but machine dependent), then the problem of minimizing the makespan can be solved in polynomial time for *any* arbitrary number of machines. Among other research papers that belong to this stream one can find the papers by Lee and Chen [54], Lee and Strusevich [55], and Tang and Liu [88].

- *Variant 2*: The variant where there is a limited buffer between the machines, a single robot that serves the entire set of m machines, and the objective is to minimize the makespan. Polynomial time procedures for several special cases of this variant appear in Panwalkar [73], Levner *et al.* [60], Kise *et al.* [49], and Stern and Vinter [84].

- *Variant 3*: The variant where there is a sufficient number of robots with no technological constraints such that any job that is finished on any one of the m machines is immediately transferred to the buffer beside the consecutive machine with no additional delays. In this case, the transportation times can be viewed as simple time-lags (delays) between any two consecutive operations of the same job. If the time-lags are all equal, then the two-machine case can be solved in polynomial time by using the well-known Johnson's algorithm for solving the $F2\,||\,C_{\max}$ problem. However, if time-lags are job-dependent, the two-machine case becomes strongly $\mathcal{NP}$-hard even for cases where either all processing times are equal to unity or time-lags have only two possible values (see Yu [92]). Another paper dealing with flow-shop scheduling and time-lags is that of Dell'Amico [22], who offers a 2-approximation algorithm for solving the two-machine case. Karuno and Nagamochi [42] improved the approximation result by Dell'Amico by constructing an (11/6)-approximation algorithm for the same problem.

- *Variant 4*: The variant of cyclic flow-shop scheduling problem in a robotic cell with a no-wait restriction. According to Che *et al.* [18], cyclic schedules can be distinguished by the number of parts entering and leaving the system in every cycle. In an $\acute{r}$-cyclic schedule, exactly $\acute{r}$ parts enter and leave the cell during each cycle. The mean cycle time of an $\acute{r}$-cyclic is defined by the cycle time divided by $\acute{r}$. The throughput rate is then the inverse of the cycle time. The objective in this variant is to maximize the throughput. Agnetis [4] provided an $O(n \log n)$ time procedure to find the optimal 1-cyclic schedule for the case where there are two-machines and a single robot. Also, Levner *et al.* [61] provided an $O(m^3 \log m)$ procedure to find the optimal 1-cyclic schedule where there are identical jobs on m machines. For identical jobs, Che *et al.* [14] and Chu [21] provided a polynomial time algorithm to solve the 2-cyclic problem, where exact algorithms to find the optimal $\acute{r}$-cyclic schedule have been proposed by Kats *et al.* [45] and Che *et al.* [13]. For a recent survey on cyclic scheduling we refer the reader to Levner *et al.* [62].

2.1.2 Problems with multiple job transferring robots

As far as we know, problems with multiple robots have only been considered for the case of cyclic scheduling in a robotic cell with no-wait restrictions. Karaznov and Livshits [43], Kats and Levner [44] and [46], and Che and Chu [15] and [17] consider the case where the robots move on parallel tracks, thus avoiding collisions between the robots. Karaznov and Livshits [43] treated the number of robots running on parallel tracks as a decision variable. They analyze the problem of minimizing the number of robots for a given periodic (cyclic) schedule. They reduced the problem to a special assignment problem that is solvable in polynomial time. Orlin [72] considered a similar periodic scheduling problem and derived another polynomial time algorithm based on minimum-cost network flows. Kats and Levner [44] presented an $O(n^5)$ time optimization algorithm to find the minimum number of robots required for all possible cycle lengths. In [46], they also developed an $O(n^3 \log n)$ optimization algorithm to find the minimum cycle time for multi-robot schedules, where the assignment of machines to robots was fixed a priori.

Che and Chu [16] and Leung and Levner [59] consider a system where all robots share the same track. Che and Chu [16] studied the problem of minimizing the cycle time, or equivalently of maximizing the production throughput, for a given number of robots, while Leung and Levner [59] provide a complete bicriteria analysis of the problem with a single robot type. In this thesis, we study a problem that is similar to the one studied in [59].

However, in contrast to the study by Leung and Levner [59], we assume that there are several types of robots distinguished by different travel times and costs. Moreover, our aim is to minimize the makespan while Leung and Levner [59] consider cyclic scheduling; thus their objective is to minimize cycle time.

Flow shop robotic scheduling problem with two transfer robots with the objective to minimize makespan, was studied recently by Lamoudan *et al.* [52]. The robots were positioned on the two sides of the machines to move finished jobs between machines by parallel movements. Each machine has a limited input/output buffer and $C(IB_i) = C(OB_i) = 1$, so when buffer is full the job has to wait on its current machine and this machine is blocked for other jobs (at least until a buffer unit becomes available). They proposed an algorithm based on meth-heuristic procedure *ant colony* (ACO) and showed that for problems without transportation times, it was able to obtain results better than other meth-heuristic procedures, such as *genetic algorithm* (GA). For problems with transportation times, results of the algorithm were compared to best results achieved for the same problems but without transportation time. In contrast to the study of Lamoudan *et al.* [52], we assume that robots are non-identical, distinguished by different transfer times and empty return times (which are non-negligible) and the input/output buffer of each machine is unlimited.

2.2 Implementations of *RL* within scheduling

In this section we provide a literature review of implementations of RL for solving flow-shop and job-shop scheduling problems.

Yusuke and Taketoshi [94] studied the feasibility of applying RL to solve the $Fm\,||\,C_{\max}$ problem for $m = 2$ and $m = 3$. Although the $F2\,||\,C_{\max}$ problem is solvable in $O(n \log n)$ time [39], Yusuke and Taketoshi applied the RL method to construct an algorithm that can solve this problem in order to compare the RL results to the optimal solution. This was used later to justify the use of this technique to solve the strongly $\mathcal{NP}$-hard $F3\,||\,C_{\max}$ problem ([27]) by using a similar method. In their research, Yusuke and Taketoshi formulate the $F2\,||\,C_{\max}$ and the $F3\,||\,C_{\max}$ problems as RL problems by applying the $TD(\lambda)$ algorithm. In their formulation the agent is simply the job dispatcher (i.e., the scheduler) of the manufacturing line, and an action simply decides which job to schedule next from a predefined set of candidates. They consider three ways to define the set of candidates and seven different ways to define the states. Moreover, they define the reward as the negative of the makespan value. By performing an experimental study, they show that for the $F2\,||\,C_{\max}$ problem the best combination of action-state is the one that (i) uses a set of candidates that include only two jobs; and (ii) defines the state as the ratio of jobs in the queue with the processing time on the first machine longer than on the second machine. We note that the two jobs in the set of candidates are defined as the ones with the minimum processing time on the first machine and the one with the maximum processing time on the second machine among all jobs in the queue (i.e., among all jobs that haven't been scheduled yet). Yusuke and Taketoshi [94] obtain similar results also for the $F3 \,||\, C_{\max}$ problem, and provide the following two insights with respect to the feasibility of applying *RL* to solve the $Fm \,||\, C_{\max}$ problem: (*i*) a good formulation can sometimes lead an agent to acquire the optimal rule that minimizes an objective function; (*ii*) an agent can learn and obtain the improved schedules even when the formulation is not perfect.

Stefn [76] used an *RL* procedure based on a Q-learning algorithm to provide a solution for the flow-shop problem on m machines with the objective to minimize the sum of machine idle times. In their formulation of the scheduling problem as an *RL* problem, they define states as job sequences, or more precisely job precedence relations. State-changes (or actions) have been defined as changes in relations. An action step is performed by a permutation operator, which sets up a job sequence according to precedence preferences. At the beginning no preferences are given, so states are traversed randomly. As learning proceeds, preferences are updated, which, in turn, influences action selection policy converging to the found quasi-optimal job sequence. Stefn [76] proposed two algorithms. The first is based on *RL* while the other combines *RL* with a *simulated annealing* (*SA*) procedure. He [76] reported the following important points that were revealed during the study: (*i*) *RL* can be combined with another heuristic procedure (such as *SA*); (*ii*) an *RL* procedure that is based on a Q-learning algorithm is adaptable and can include more elements; and (*iii*) *RL* can deal with large scale scheduling problems and provide a near-optimal solution. Further research by Zhang *et al.* [95] used an *RL* procedure based on a *TD*(λ) algorithm to solve the $Fm \,||\, C_{\max}$ problem. To do that they first convert the flow-shop scheduling problem into a *semi-Markov decision process* (*SMDP*) problem by constructing elaborate state features, actions, and a reward function. The conversion was done in such a way that minimizing the accumulated reward in the converted problem is equivalent to minimizing the schedule objective function. To examine the performance of their proposed *RL* algorithm in comparison with other scheduling methods, they conducted a set of computational experiments on benchmarking problems. Their results supported the efficiency of the proposed algorithm and illustrated that the *RL* approach is a promising computational approach for solving flow-shop scheduling problems.

The *RL* method was employed by Aydin and Oztemel [8] and Wei and Zhao [90] to also solve scheduling problems in a dynamic job-shop scheduling environment, where jobs may arrive at the shop at any time. They developed an adaptive method of rules selection for dynamic job-shop scheduling using *RL* where the scheduler does not possess detailed information about the jobs, which may arrive at the shop at any time. The problem presented in Aydin and Oztemel [8] is to schedule a set of jobs (each job consists of a set of operations), under a set of constraints, in terms of a certain criteria. For this, they developed an improved version of *RL* algorithm based on *Q*-learning, which they refer to as *Q-III*. The agent was trained by the *Q-III* through the learning stage, and then it had to make decisions in order to schedule operations. In other words, the agent selects in real-time the most appropriate rule for schedule operations according to the shop conditions, and then the simulated environment performs the scheduling by the rule that was selected. They found out, at the end of training, that the agent provides better results than the traditional alternatives, such as using the *SPT* rule. In Wei and Zhao [90] the agent was trained by a *Q*-learning algorithm, which enables it to select the appropriate rule in real-time. The rule is used for selecting a job from the buffer, when the decision on which rule to use is based on the status of the system buffer. The goal of their research was to minimize mean tardiness and through various sets of examples they were able to show that the *Q*-learning algorithm has superiority over most of the conventional rules compared. Wei and Zhao [91] extend their work in [90] to consider both machine and job selections.

Martinez *et al.* [68] study a static flexible job-shop scheduling problem with the objective of minimizing the makespan. They presented an improved *RL* approach that combines both learning and optimization to solve the problem. By conducting an experimental study they compare the quality of their algorithm to other commonly used meth-heuristic procedures such as ones that are based on *GA*, *ACO*, and *Tabu search (TS)*, and showed that their proposed algorithm provides the smallest average gap from the best known lower bound value.

2.3 Collaboration in *RL*

Two important surveys by [23] and [67] on the field of *MARL* were conducted in the last period, pointing out difficulties and challenges. "With the ever increasing interests in theoretical researches and practical applications, currently there have been a lot of efforts towards providing some solutions to this challenge. However, there are still many difficulties in scaling up the *MARL* to *MRS*" [23]. The main difference between single robot and *MAS* is the environment, where in *MAS* other adapting agents make the environment non-stationary. As a result, the Markov property that a traditional learning agent relies upon is violated. However, this fact is usually neglected and the basic assumption of *Q*-learning is violated and some researchers apply *Q*-learning in *MRS* [23]. In addition, for the single robot, the reward that the learning robot receives depends on its own actions, while rewards that learning robots receive depend not only on their own actions but also on the actions of other robots.

"The distributed autonomous robotic system has superiority of robustness and adaptability to dynamical environment, however, the system requires the cooperative behavior mutually for optimality of the system. The acquisition of action by *RL* is known as one of the approaches when the multi-robot works with cooperation mutually for a complex task" ([71]). Their research deals with transporting problem in *MAS* using *Q*-learning algorithm, and aims to establish effective cooperation. Tomofumi *et al.* [71] define different rewards that can be characterized as one of the following: positive for making transfer or positive/negative for being cooperative/non-cooperative, respectively. The *Q*-learning algorithm performed the least number of cooperations and in the process small (compared to other methods) total number of unnecessary encounters. In conclusion, Tomofumi *et al.* remark that it is difficult to deal with cooperation but effective. Research by Laetitia *et al.* [51] focuses on decentralized *RL* in cooperative *MAS*, where a team of independent learning robots try to coordinate their individual behavior to reach a coherent joint behavior. In other words, the agents share the same goal and the common payoff can be jointly maximized. Despite difficulties mentioned above, Laetitia *et al.* [51] have presented a decentralized *RL* algorithm, based on *Q*-learning, for independent learners that computes a better policy in a cooperative *MAS* without additional information or communication between agents than existing algorithms. Later research of Dahl *et al.* [19] dealt with modeling the effects of robot interaction in multi-robot systems, i.e., group dynamic. They have presented a cooperative *RL* algorithm, based on *Q*-learning, in order to solve a multi-robot task allocation problem, where each individual robot handles traditional *RL* algorithm with actions, state and *Q*-values. The robots can can get in each other way, and hence to optimize performance, robots must find the correct balance between levels of interference. They show that when greedy individuals continually update local *Q*-values, achieved solutions are optimal. So, ordinary decentralized *Q*-learning could be applied to some cooperative multi-agent systems.

In addition to *Q*-learning, Erfu and Dongbing [23] present and describe different *RL* algorithms from the literature that were implemented by using the *Q*-learning, *SARSA*, and *SARSA*(λ) techniques. Yet, they point out that "Although there have been a variety of *RL* techniques that are developed for multi-agent learning systems, very few of these techniques scale well to *MRS*. On the one hand, the theory itself on *MARL* systems in the finite discrete domains are still underway and have not been well established." Later study by Lucian *et al.* [67] on *MARL* notes that several new challenges arise for *RL* in *MAS*, including the difficulty of specifying a learning goal; the non-stationarity of the learning problem; increasing number of learning agents (i.e., the exponential growth of state-action spaces); and the need for coordination as each learning agent must follow the other learning agents and so must be able to coordinate its behavior with theirs (see also [20] and [47]).

Collaborative RL was first introduced on a human-robot collaboration system by Kartun *et al.* [41]. They propose $CQ(\lambda)$ algorithm which is based on $Q(\lambda)$. The "$CQ(\lambda)$ included two collaboration levels: (i) Autonomous (AO) - the robot decides which actions to take according to its learning function, and (ii) Semi-autonomous (SAO) - a human operator guides the robot and the robot combines this knowledge into its learning function" ([41]). The robot was aware of its learning performance and was able to switch from autonomous to semi-autonomous and ask for human assistance. The $CQ(\lambda)$ integrates the experience of a robot and a human, and so accelerates learning, i.e., reduces the long learning times of the $Q(\lambda)$. Future research of this work followed from the question of "How can the robot be sure that all human operator suggestions are beneficial? It may be the case that some human advice does not contribute to the robot learning process" [41]. Thus, research by Gil *et al.* ([30]) improved the $CQ(\lambda)$ algorithm and studied the system when the robot was able to decide whether to accept or reject the advice, i.e., they provide the robot with a tool to estimate the quality of each advice. However, one challenge remains open: how the robot can assess human expertise, based on its past advises, which will enable it to decide whether to continue to collaborate with the expert or not.

2.4 Research gaps and directions

In this research we analyze a set of single and multi-robot flow-shop scheduling problems with the scheduling objective of minimizing the makespan. Our research aims to provide a theoretical analysis of various scheduling problems, which were not analyzed in the literature yet, and to introduce a new RL techniques for solving hard robotic scheduling problems. The following lists the gaps we identified in the literature review and the corresponding research directions.

1. Scheduling models with job-independent processing time addresses one of the most important and widely studied special cases in the scheduling literature. These models deal with the real life applications of repetitive manufacture of the same product. When reviewing the literature related to flow-shop problems with a single robot, we found out that no one studied the *Variant 1* (see Section 2.1.1) of the problem with job-independent (but machine-dependent) processing times. Thus, one of our research directions is to construct algorithms to solve this problem.

2. The literature review reveals the difficulty of analyzing flow-shop scheduling problems with multiple job transferring robots. In fact, as far as we know, all studies related

to this field that provide theoretical analysis assume cyclic production of identical jobs (see, e.g., Karaznov and Livshits [43], Kats and Levner [44], and [46], Che and Chu [15], and [17], and Leung and Levner [59]). However, the more difficult non-cyclic production is overlooked. Thus, another direction is to analyze the non-cyclic version of the problem involving multiple transferring robots. Accordingly, we define an objective function to minimize the makespan rather than to minimize the cycle time.

3. Che and Chu [16] have pointed out that an important aspect that must be taken under consideration when dealing with *MRS* is the number of robots and their type, which significantly effects production costs. However, as far as we know, all multi-robot flow-shop scheduling problems analyzed in the literature, used a single robot type, and in most cases the number of robots from the same type is given in advance. Thus, one of our main directions is to study a robotic flow-shop scheduling problem, where there are several robot types and both the robot selection and their assignment to machines are decision variables.

4. *RL* is one of the most important methods of machine learning. It is widely used to solve many problems in a large number of fields. Yet, it is not widely used in the field of robotic scheduling. In fact, as far as we know, collaborative *RL* (introduced by [41]) has not been applied yet for the solution of hard robotic flow-shop problems. Thus, one of our main research directions is to apply collaborative *RL* on various robotic scheduling problems in order to enable faster results (accelerate learning) and to examine the quality of the obtained results.

5. When applying the *RL* method in the field of scheduling, researchers have used single agent models (see, e.g., Yusuke and Taketoshi [94], Stefn [76], and Zhang *et al.* [95]). However, in many cases faster and better results can be achieved when using a collaborative multi-agent system. Thus, we aim to construct *RL* models that can efficiently coordinate between the agents. Moreover, we aim to implement the collaborative *RL* models with different levels of collaboration between the agents. This will enable us to examine the effect of the collaboration level on the quality of the obtained solution.

3 Problems Definitions, Research Objectives, Significance and Thesis Organization

3.1 Problems definitions

In section 2.4 we provide our motivation, which is derived from existing gaps in the literature. We aim to close some of the gaps provided, by analyzing a set of three robotic scheduling problems that haven't been studied before. In each of these problems we have to schedule a set of n jobs, $\mathcal{J} = \{J_1, ..., J_n\}$, which is available for processing at time zero and has to be scheduled on a set of m machines, $\mathcal{M} = \{M_1, ..., M_m\}$, in a flow-shop scheduling system. Moreover, there are Q types of robots, where the cost of a single robot of type q is δ_q for $q = 1, ..., Q$. Moreover, in all problems t_{iq} represents the transportation time required for a robot of type q to transfer a job from machine M_i to machine M_{i+1} (we assume that this time is job-independent), and te_{iq} represents the time required for a robot of type q to move empty (without carrying a job) from machine M_{i+1} to machine M_i. In any special case where there is a single robot type, we omit the q index, such that, for example t_i represents the transportation time required for the robot to transfer a job from machine M_i to machine M_{i+1}. Next, we provide the additional information required to provide a formal definition for each of the three problems we study.

3.1.1 Problem 1 - Scheduling identical jobs with a single robot

In *Problem 1* we focus on the special case of the robotic scheduling problem that consists of a single robot, and where job processing times are job-independent (and may either be machine-independent or machine-dependent). We focus on *Variant 1* of the problem (see Section 2.1.1), where there is an unlimited buffer between the machines and the objective is to minimize the makespan. However, since for the three machine case, we prove that the optimal schedule does not exploit the capacity of the buffer, our results (for the three machine case) holds also for the more restrictive cases where the buffer size is limited. Figure 1 below illustrates our robotic flow-shop problem with a single transferring robot and identical processing times.

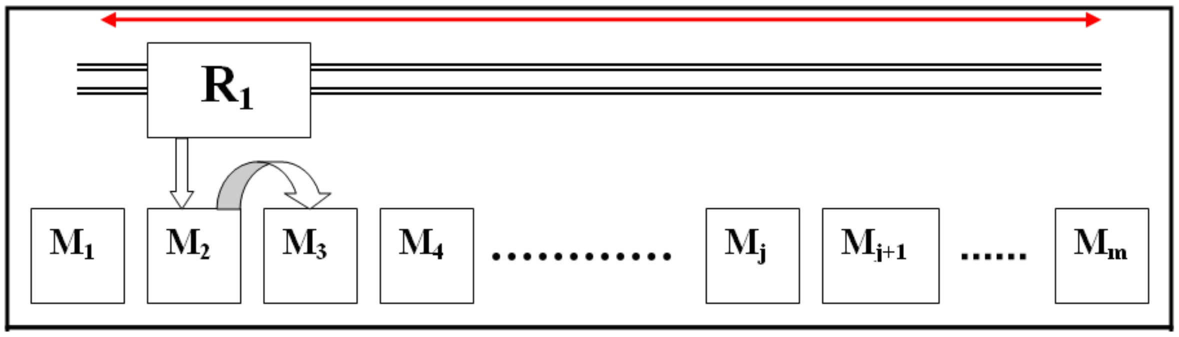

Figure 1: An illustration of single robot job-machine scheduling system.

3.1.2 Problem 2 - The *Robot Selection and Scheduling Problem* ($\mathcal{RSSP}$)

In *Problem 2*, which we refer to as the *Robot Selection and Scheduling Problem* ($\mathcal{RSSP}$), we take the challenge of combining the two important decisions of robot selection and scheduling into the same unified decision-making model. The aim is to provide the scheduler with tools to optimally coordinate these two important decisions in a non-cyclic scheduling environment, where the objectives are to minimize the makespan and the total robot selection cost. Thus, in contrast to most models where the number of robots and their type is predefined to the scheduler, we study a problem where the scheduler can select both the number of robots to be used and their type. More particularly, we assume that there are Q types of robots, where the cost of a single robot of type q is δ_q for $q = 1, ..., Q$, and each robot type has its own transportation abilities, i.e., both transportation times and empty return times are robot-type dependent.

In our $\mathcal{RSSP}$, we consider the case where the robots move on an identical track positioned alongside a machine transfer line. Due to the limited working space envelopes of the robots and to avoid collisions, each robot is assigned to a portion of the track and performs job transfers between all reachable machines from its assigned portion of the track. Let k be the number of robots serving the system and let $\mathcal{M}_r = \{M_{l_r}, ..., M_{l_{r+1}}\}$ be a subset of consecutive machines designated to robot R_r for $r = 1, ..., k$ with $l_1 \overset{\text{def}}{=} 1$ and $l_{k+1} \overset{\text{def}}{=} m$. Robot R_r is responsible for transferring jobs between successive machines in set $\mathcal{M}_r$. Note that $\mathcal{M}_r \cap \mathcal{M}_{r+1} = M_{l_{r+1}}$ and robot R_r is responsible for transfers to machine $M_{l_{r+1}}$, while robot R_{r+1} is responsible for transfers from machine $M_{l_{r+1}}$for $r = 1, ..., k-1$. Figure 2 below illustrates the flow-shop system and its partition into k working space envelopes of the robots.

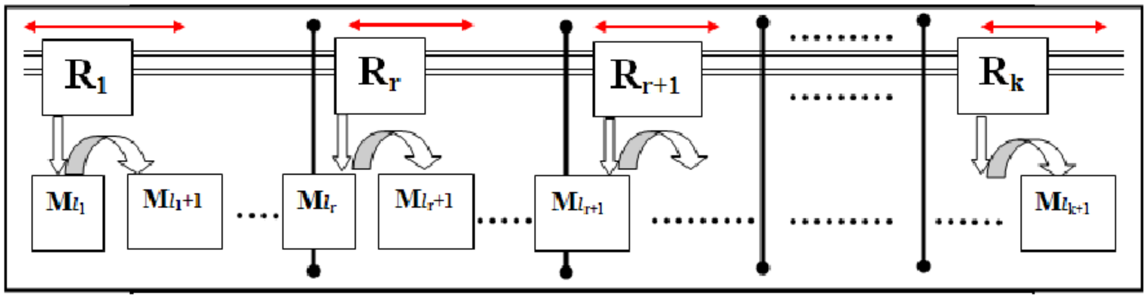

Figure 2: An illustration of the robot job-machine scheduling system.

We assume that processing times are both job and machine independent (that is, $p_{ij} = p$ for $j = 1, ..., n$ and $i = 1, ..., m$). Thus, the job scheduling problem is redundant (i.e., any permutation schedule is optimal) and a solution to the problem $\mathcal{RSSP}$ includes the following two parts: (a) robot selection and (b) robot scheduling. The robot selection part comprises of a selection of an ordered list of k robots $\{R_1, R_2, ..., R_k\}$ (where k itself is a decision variable) from the set of Q robot types (we may select more than a single robot of the same type) and from the assignment of a subset of machines, $\mathcal{M}_r = \{M_{l_r}, ..., M_{l_{r+1}}\}$, to each robot R_r for $r = 1, ..., k$ such that $\mathcal{M}_1 \cup \mathcal{M}_2 \cup ... \cup \mathcal{M}_k = \{M_1, ..., M_m\}$. The robot scheduling part defines a set of *moves* for each robot that indicates the sequence in which the robot serves the machines. Similar to Che and Chu [17], a solution to the scheduling part is *feasible* if it obeys the following two restrictions:

Restriction 1: (no-wait restriction): Jobs are not allowed to wait between two consecutive machines; that is, once a job has finished its processing on machine M_i it must be immediately transferred to machine M_{i+1} for $i = 1, ..., m - 1$.

Restriction 2: (no machine idle time): Once a machine has started work, it must process the entire set of n jobs consecutively.

We evaluate the quality of a feasible solution to $\mathcal{RSSP}$ by two different criteria (performance measures). The first is the makespan criterion denoted by $C_{\max} = C_n$, and defined by the completion time of the last job (job J_n) on the last machine (machine M_m). The second is the total cost of the assigned robots defined by

$$TRC = \sum_{r=1}^{k} \delta_{\{r\}}$$

where $\{r\}$ is the type of robot R_r.

In any multi-criteria problem it is very important to point out the nature of the optimization being performed as different criteria are often conflicting. Here we are interested in solving the following four variations of the $\mathcal{RSSP}$:

1. $\mathcal{RSSP}_1$: Find a feasible solution which minimizes the total integrated cost, i.e., $C_{\max} + TRC$.

2. $\mathcal{RSSP}_2$: Find a feasible solution which minimizes $C_{\max}$ subject to $TRC \leq \overline{TRC}$, where $\overline{TRC}$ is a given upper bound on the total robot assignment cost.

3. $\mathcal{RSSP}_3$: Find a feasible solution which minimizes TRC subject to $C_{\max} \leq \overline{C}_{\max}$, where $\overline{C}_{\max}$ is a given upper bound on the makespan value.

4. $\mathcal{RSSP}_4$: Identify a Pareto-optimal solution for each Pareto-optimal point, where a feasible solution S is called *Pareto-optimal* (*non-dominated* or *efficient*) with respect to criteria $C_{\max}$ and TRC; if there does not exist another feasible solution S' such that $C_{\max}(S') \leq C_{\max}(S)$ and $TRC(S') \leq TRC(S)$, with at least one of these inequalities being strict.

Note that solving problem $\mathcal{RSSP}_4$ also solves problems $\mathcal{RSSP}_1$-$\mathcal{RSSP}_3$ as a by-product. Note also that the decision version (DV) of problem $\mathcal{RSSP}_2$ is identical to that of problem $\mathcal{RSSP}_3$, and is defined below:

Definition 21 DV: *Given parameters $\overline{C}_{\max}$ and $\overline{TRC}$, determine whether there is a solution for the $\mathcal{RSSP}$ with $C_{\max}(S) \leq \overline{C}_{\max}$ and $TRC(S) \leq \overline{TRC}$.*

The fact that both the $\mathcal{RSSP}_2$ and the $\mathcal{RSSP}_3$ problems share the same decision version implies that either both or neither of them is $\mathcal{NP}$-hard.

3.1.3 Problem 3 - Scheduling identical jobs with two robots on parallel tracks

Problem 3 is a variant of the robotic flow-shop scheduling problem, where two robots (which may be of a different type) moving on parallel tracks are serving the production line. However, in contrast to *Problem 2*, we assume that the robots and their type are predefined to the scheduler. Since the robots are working on parallel tracks there is no risk of collisions, meaning that the robots are independent and each can work on any machine in set $\mathcal{M}$. Furthermore, since robots' tracks are located on opposite sides, the two robots can transfer different jobs from and to the same buffer at the same time. Figure 3 below illustrates our robotic flow-shop problem with two robots on parallel tracks.

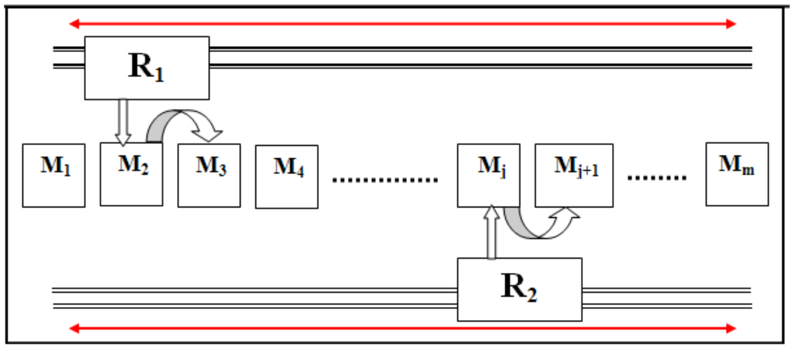

Figure 3: An illustration of our scheduling system with two robots on parallel tracks.

The objective in *Problem 3* is to find a schedule for both robots and a job schedule on the machines that all together minimizes the makespan. We assume that the capacity of all buffers is unbounded, and that job processing times are job-independent (and may either be machine-independent or machine-dependent).

3.2 Research objectives

Our research objective is to analyze each of the three problems presented above, and to provide tools for optimization. More particularly, our objectives are to:

1. Provide tight lower bounds for the corresponding scheduling problems. These bounds will be used either for proving that a specific algorithm provides the optimal solution, or for comparing the performance of a suggested heuristic algorithm.

2. Provide polynomial time procedures, when possible, to various variants of the three problems presented above. When we couldn't provide such algorithms we will attempt to prove that the corresponding problem is $\mathcal{NP}$-hard.

3. Try to provide both exact algorithms (such as pseudo polynomial time algorithms) and approximation algorithms (such as an *FPTAS*) for problems in which polynomial time procedure is not available. Moreover, we will focus on applying collaborative *RL* methods in order to find a near optimal solution.

4. Show that collaborative *RL* methods can be used to obtain good solutions for scheduling problems in general and for robotic flow-shop scheduling problems in particular.

5. Develop collaborative *RL* models that are based on collaboration between a robot and an adviser. The tools will provide the robot with the ability to assess the adviser's

level of expertise, which in turn enables it to make wise decisions on whether to use the adviser or not. Note that current models have either assumed that the adviser is indeed expert, and thus the robot has to take his advises (see [41]) or use a learning process that is based on the quality of each advice rather the quality of the adviser (see Gil *et al.* ([30]).

6. Apply collaborative *RL* with different levels of collaboration on various *multi*-robotic scheduling problems, and to examine the benefit result in from increasing collaboration levels (in terms of the solution value).

3.3 Research significance and main results obtained

According to Lebacque and Brauner [53], "Global competition has increased the need of automation in production cells and computer-controlled material handling devices have flourished over the recent years." Furthermore, since *MRS* can often be used to fulfill tasks that are very difficult and complex to be accomplished by an individual robot ([23]), and since generalizations of scheduling problems within *MRS* is a problem that hasn't been dealt with seriously ([38]), it is very important to deal with and analyze flow-shop scheduling problems within *MRS*. Thus, we focus on analyzing three different robotic flow-shop scheduling problems. The research significance and main results for each of the problems we analyzed is summarized below:

1. **Analysis and solution procedures for the robotic flow–shop scheduling problem with identical jobs and a single robot** - Various scheduling problems with a single material handling device (robot) were studied in the literature (see Lebacque and Brauner [53] for a survey on this field up to 2008), with some problems remaining open ([38]). One of them, to the best of our knowledge, is the flow-shop scheduling problem with a single robot and job-independent processing times (see *Problem 1* in Section 3). This problem has a lot of real life applications in production systems that produce identical products in a flow-shop scheduling system. Thus, in this thesis *we analyze Problem 1, and provide a closed form solution for the three-machine case.* Moreover, for the more general case, where m is arbitrary, *we use a collaborative RL formulation for providing a near-optimal solution.* In our collaborative *RL* formulation, the robot, while learning, collaborates with another entity, the adviser, and together they aim to provide the best possible solution for the scheduling problem. The robot in our model uses the ε-greedy method for action selection, while the adviser uses the softmax action

selection method. The robot, when deciding that the learning process is ineffective, asks for assistance (advice) from the adviser. When receiving the advice, the robot examines the quality of the advice and acts accordingly. If the robot learns that the adviser is ineffective it decides to stop the collaboration; if the adviser is found to be effective, the robot decides to continue the collaboration with the adviser. The adviser, when called, assists the robot and so provides it an advice, i.e., an entire scheduling for a given problem. Such *a collaborative RL formulation is unique and has not been applied yet for solving a scheduling problem.* Note that current collaborative *RL* models, which include collaboration between an agent and an adviser, have either assumed that the adviser is expert, and thus the robot has to take his advice (see [41]) or used a learning process that is based on the quality of each advice (see Gil *et al.* ([30]). We, on the other hand, use a learning process that assesses the quality of the adviser.

2. **Analysis and a solution procedure for the** $\mathcal{RSSP}$ (*Problem 2*) - In systems where tasks are too difficult and too complicated to be accomplished by a single robot, the need for more robots arises. An important aspect that must be taken under consideration when dealing with *MRS* is the number of robots ([16]) and their type, which significantly affects production costs.

   Our model of the $\mathcal{RSSP}$ *includes some new features that have been discussed rarely or not at all in the literature.* Among those features are several robot types, an arbitrary number of machines, a possibility to control the number of robots assigned to the production line and their assignment to machine sets, and a bicriteria approach for analysis. We believe this is the first time that this type of problem has been treated in the literature, and addresses a very important problem in multiple robotic systems operation. On the other hand, to be able to provide a mathematical based analysis we used the following assumptions; (*i*) There are no-wait and no-idle restrictions (see Restrictions 1-2 in the definition of the $\mathcal{RSSP}$); (*ii*) Transportation times are job-independent; and (*iii*) Job processing times are *all* equal. The no-wait and no-idle restrictions are well justified by many real-life applications (see, e.g., Che and Chu [17], Hall and Sriskandarajah [36], Saadani *et al.* [81], and Goncharov and Sevastyanov [32]) and are widely used in the flow-shop scheduling literature (see Adiri and Pohoryles [1], Kalczynski and Kamburowski [40], Ruiz *et al.* [80], Lei and Wang [56], Levner *et al.* [61], Agnetis [4], Agnetis and Pacciarelli [5], Che *et al.* [13], and [14], Leung and Zhang [57], Leung *et al.* [58], and Levner *et al.* [62] among many others). Even the assumption of job-independent (but machine-dependent) transportation times can be easily justified and thus are commonly used in robotic flow-shop scheduling problems (see, e.g., Kise *et al.* [50], Levner *et al.* [61], Agnetis [4], Agnetis and Pacciarelli [5], Che *et al.* [13], Che *et al.* [14], Leung and Zhang [57], Leung *et al.* [58], Leung and Levner [59] and Levner *et al.* [62]). However, the third assumption of equal processing times seems to be the most restrictive one. Nevertheless, this assumption is crucial for providing a mathematical based analysis of the $\mathcal{RSSP}$ problem. This is well justified by the fact that no one yet was able to solve the very restricted model where there are m machines, a single robot, and job-independent (but machine-dependent) processing times without including additional simplifying assumptions (such as cyclic production). Note also that the third assumption, although very restrictive, was used by Hurink and Knust [38] in a large portion of their paper and was the most crucial assumption that enabled exact solution procedures. Moreover, this assumption was used by many other researchers who dealt with special cases of flow-shop scheduling problems (see, e.g., Adiri and Amit [2], Yu *et al.* [93], Mosheiov *et al.* [69], Mosheiov and Oron [70], Lim *et al.* [63], Ageev and Baburin [3], and Brucker and Shakhlevich [11]).

The main results obtained in this thesis for the $\mathcal{RSSP}$ problem includes:

- A polynomial reduction of the $\mathcal{RSSP}$ to a bicriteria shortest path problem in an acyclic graph.
- A polynomial time procedure to solve the $\mathcal{RSSP}_1$.
- A proof that $\mathcal{RSSP}_2 - \mathcal{RSSP}_4$ are $\mathcal{NP}$-hard.
- A proof that problems $\mathcal{RSSP}_2 - \mathcal{RSSP}_4$ are solvable in pseudo-polynomial time, and that there exists an *FPTAS* for the solution of $\mathcal{RSSP}_2$ and $\mathcal{RSSP}_3$.
- The construction of polynomial time procedures to solve some important special cases of the $\mathcal{NP}$-hard problems, including: (i) the case of robot-independent transportation times; and (ii) the case of identical robot costs.

3. **Scheduling identical jobs with two robots on parallel tracks** (*Problem 3*) - Since a polynomial time procedure to solve this problem does not exist, we use a novel collaborative *RL* formulation for the problem that is used to obtain a near-optimal solution. We study the following new four different levels of collaboration between robots: (a) *Full* - robots are working together, performing both self- and joint-learning and sharing full information; (b) *Pull* - only one robot can decide when and if to learn from the other robot; (c) *Push* - one robot may force the other robot when and how to learn from it; and (d) *None* - each robot is autonomous, i.e., robots do not share information and each learns independently. *For each level of collaboration we were able to*

formulate the scheduling problem as a collaborative RL problem overcoming two known difficulties within RL (the non-stationary environment and coordination between the agents [23], [51], and [67]), and to determine the benefit of using each level of collaboration in terms of the quality of our solution.

In addition to providing methods for solving our problems, we want to emphasize that, as far as we know, our study is *the first to implement a collaborative RL method* (using Q-learning) for solving robotic scheduling problems, and the first to study the benefit of using increasing levels of collaboration between agents (either two robots or a robot with an adviser) in such an environment.

3.4 Thesis organization

The rest of the thesis is organized as follows. In the following sections, we analyze each of *Problems 1-3*, first presenting theoretical analysis for *Problem 1* (see Section 4) and *Problem 2* (see Section 5), followed by collaborative *RL* Robot-Adviser (*RA*) algorithm for *Problem 1* (see Section 6), and collaborative *RL* Robot-Robot (*RR*) algorithm for *Problem 3* (see Section 7). Finally, in Section 8, a summary and future research concludes the thesis.

4 Polynomial Time Procedure to Solve *Problem 1* with Three Machines

This section deals with *Problem 1* and presents a polynomial time procedure to solve the problem with three machines, i.e., to solve the $F3, R1|p_{ij} = p_i, t_{ij} = t_i|C_{\max}$ problem.[1] This is done by decomposing the problem to a set of sub-problems, and providing an optimal schedule for each sub-problem separately. The optimality of each schedule (for each sub-problem) is proven by showing that the makespan value obtained by the suggested schedule matches the lower bound value. Next, we present our lower bound value, the decomposing of the problem into four sub-problems, and necessary and sufficient conditions for feasibility of a schedule.

4.1 Schedule definition in terms of robot moves

Since the jobs in our $F3, R1|p_{ij} = p_i, t_{ij} = t_i|C_{\max}$ problem are identical, we can assume, without loss of generality, that the jobs are scheduled in the sequence of their indices on each of the three machines. Thus, our objective is to determine the optimal robot schedule that minimizes the makespan. A robot schedule can be easily defined by a sequence of $4n-2$ robot moves. These $4n-2$ moves includes (i) n moves, each of a different job from M_1 to M_2; (ii) n moves, each of a different job from M_2 to M_3; (iii) $n-1$ empty moves from M_3 to M_2; and (iv) $n-1$ empty moves from M_2 to M_1. Let $Z = (z_1, ..., z_{4n-2})$ be a sequence of $4n-2$ robot moves, where $z_\lambda = i$ implies that the λ'th move of the robot is a non-empty move from machine M_i to machine M_{i+1}, and $z_\lambda = e_i$ implies that the λ'th move of the robot is an empty move from machine M_{i+1} to machine M_i. Given a sequence of robot moves, we construct a feasible schedule according to Algorithm 1 below.

Algorithm 1 *Constructing a feasible schedule for a given sequence of robot moves.*

- *Schedule the jobs on* M_1 *one after the other with no delays, such that job* J_j *is scheduled during time interval* $((j-1)p_1, jp_1]$ *for* $j = 1, ..., n$.
- *Once a robot finishes a move, if the next move is an empty move* (e_i *for* $i = 1, 2$) *it is immediately done. Otherwise, if the move is a non-empty move* (i *for* $i = 1, 2$), *the*

[1] The analysis presented in this chapter was recently submitted to the International Journal of Production Research.

robot moves a job as early as possible (once a job is available at the output buffer of M_i).

- *We schedule each job J_j $(j = 1, ..., n)$ on M_i $(i = 2, 3)$ as early as possible, which is the latest time between (i) the time that J_j is downloaded at the input buffer of M_i; and (ii) the time that the M_i finishes the processing of J_j (with the completion time of J_0 on M_i being zero, by definition).*

Let us define the following subsequences of moves: $Ss_1 = (1, 2, e_2, e_1)$; $Ss_2 = (1, 2)$; $Ss_3 = (1, e_1)$; $Ss_4 = (e_2, 2)$; and $Ss_5 = (e_2, e_1)$. We will prove that for any instance of the problem there exists an optimal sequence of robot moves that is taken out of the following two sequences of moves.

- *Sequence 1*: $Z_1 = (Ss_1, Ss_1, ..., Ss_1, Ss_2)$, where the robot does not perform two non-empty consecutive moves between the same two consecutive machines. This sequence of moves is constructed from $n - 1$ subsequences of type Ss_1 followed by a single subsequence of type Ss_2. In this sequence of moves, for $j = 1, ..., n$, the robot (i) moves J_j from M_1 to M_2; (ii) waits besides M_2 for the completion of J_j; (iii) moves J_j from M_2 to M_3; and (iv) if $j < n$ returns empty to M_1.

- *Sequence 2*: $Z_2 = (Ss_1, ..., Ss_1, Ss_3, Ss_1, ..., Ss_1, Ss_2, Ss_4)$ if $y = n$ and

  $Z_2 = (Ss_1, ..., Ss_1, Ss_3, Ss_1, ..., Ss_1, Ss_2, Ss_4, Ss_5, Ss_1, ..., Ss_1, Ss_2)$ if $y < n$. Here, the robot makes two consecutive moves from M_1 to M_2 exactly once (which implies that the robot makes two consecutive moves from M_2 to M_3 exactly once as well). Let the two consecutive moves from M_1 to M_2 be done on jobs J_x and J_{x+1}, and the two consecutive moves from M_2 to M_3 be done on jobs J_{y-1} and J_y with $x \in \{1, ..., n-1\}$ and $y \in \{x+1, ..., n\}$. Then, this sequence of moves starts with $x - 1$ subsequences of type Ss_1, followed by a single subsequence of type Ss_3; a set of $y - x - 1$ subsequences of type Ss_1; a single subsequence of type Ss_2; and a single subsequence of type Ss_4. If $y = n$ the sequence of moves ends. Otherwise, the sequence continues with a single subsequence of type Ss_5; a set of $n - y - 1$ subsequences of type Ss_1; and a single subsequence of type Ss_2. Here, the robot starts with a sequence of moves where for $j = 1, ..., x - 1$ the robot moves J_j from M_1 to M_2; waits beside M_2 for the completion of J_j on M_2; moves J_j from M_2 to M_3; and returns empty to M_1. Then, the robot moves J_x from M_1 to M_2 and returns empty to M_1. After completing this move, for $j = x+1, ..., y-1$ the robot moves J_j from M_1 to M_2; J_{j-1} from M_2 to M_3; and returns empty to M_1. This set of moves is followed by a move of J_y from M_1 to M_2; a move of

J_{y-1} from M_2 to M_3; an empty return move from M_3 to M_2; and a move of J_y from M_2 to M_3. If $y = n$, the sequence of moves ends as J_n has been moved from M_2 to M_3. Otherwise, the robot returns empty from M_3 to M_1. Then, for $j = y+1, ..., n-1$, the robot moves J_j from M_1 to M_2; waits besides M_2 for the completion of job J_j on M_2; moves J_j from M_2 to M_3 and returns empty to M_1. Finally, the robot moves J_n from M_1 to M_2; waits besides M_2 for the completion of J_n on M_2; and moves J_n from M_2 to M_3.

4.2 Simple lower bounds and problem decomposition

Let us first calculate a lower bound, LB_f, which is based on the processing of all jobs on machine M_f for $f \in \{1, 2, 3\}$. The earliest time machine M_f can start processing J_1 is at $\sum_{i=1}^{f-1} p_i + \sum_{i=1}^{f-1} t_i$ (where $\sum_{i=1}^{0} p_i = \sum_{i=1}^{0} t_i = 0$ by definition). Thus, processing on machine M_f cannot be completed before $\sum_{i=1}^{f-1} p_i + \sum_{i=1}^{f-1} t_i + np_f$. After J_n is completed on M_f it has to be processed on machines $M_{f+1}, ..., M_3$. Thus, the earliest time to complete J_n on M_3, which is in fact a lower bound on the makespan value, is

$$LB_f = \sum_{i=1}^{f-1} p_i + \sum_{i=1}^{f-1} t_i + np_f + \sum_{i=f+1}^{3} p_i + \sum_{i=f}^{2} t_i = (n-1)p_f + \sum_{i=1}^{3} p_i + \sum_{i=1}^{2} t_i, \quad (3)$$

for $f = 1, 2, 3$. Let us next calculate another lower bound which is based on the robot moves. The robot cannot start its moves before J_1 is completed on M_1 at p_1. From this time point the robot has to move the n jobs from M_1 to M_2 and from M_2 to M_3, and has to return empty $n-1$ times from each machine to its predecessor machine. Thus, a lower bound for the time that the robot finish to transfer J_n to M_3 is at $p_1 + n\sum_{i=1}^{2} t_i + (n-1)\sum_{i=1}^{2} te_i$. Lastly, job J_n has to be processed on M_3, and thus the makespan value is lower bounded by

$$LB_r = p_1 + n\sum_{i=1}^{2} t_i + (n-1)\sum_{i=1}^{2} te_i + p_3. \quad (4)$$

Obviously, when several lower bound values are given, the ultimate lower bound is the maximum between them. Thus,

$$LB = \max\left\{\max_{f=1,2,3}\{LB_f\}, LB_r\right\} = \\ \sum_{i=1}^{2} t_i + \sum_{i=1}^{3} p_i + \max\left\{(n-1)p_{\max}, (n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) - p_2\right\} \quad (5)$$

provides a lower bound on the makespan value, where $p_{\max} = \max_{f=1,2,3}\{p_f\}$.

Hereafter, we refer to M_f as a *bottleneck machine* if $p_f = \max_{i=1,2,3}\{p_i\}$. Moreover, we say that machine M_f is the system's bottleneck if $p_f = \max_{i=1,2,3}\{p_i\}$ and $p_f \geq \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$, and that the robot is the system's bottleneck if $\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i \geq \max_{i=1,2,3}\{p_i\}$. Note that if machine M_f is the system's bottleneck, then $LB = LB_f$. However, if the robot is the system's bottleneck then it is not necessarily true that $LB = LB_r$.

In order to solve our problem, we decompose it into four different sub-problems (*cases*) each of which is related to a different bottleneck of the system. Case f ($f = 1, 2, 3$) is where machine M_f is the systems' bottleneck, and thus

$$p_f = \max\left\{p_1, p_2, p_3, \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right\}, \tag{6}$$

and Case 4 is where the robot is the systems' bottleneck, and thus

$$\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i = \max\left\{p_1, p_2, p_3, \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right\}. \tag{7}$$

Note that for some instances there might be an overlap between the different cases. In such a case, one has to arbitrary decide to which case the corresponding instance belongs. Below, we provide revised (tighter) lower bounds for various subcases of Cases 1, 3 and 4.

4.3 Tighter lower bounds

4.3.1 Tighter lower bounds for various subcases of Case 1

According to (5), for Case 1 where

$$p_1 = \max\left\{p_1, p_2, p_3, \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right\}, \tag{8}$$

we have that

$$LB = LB_1 = (n-1)p_1 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{3} p_i. \tag{9}$$

However, when $p_1 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ and either $t_2 + te_2 = \max\{p_2, p_3, t_2 + te_2\}$ or $p_3 = \max\{p_2, p_3, t_2 + te_2\}$, we next provide a revised (tighter) lower bound denoted by LB_1'. To do so, we consider two possible scenarios:

- Scenario (a): the robot does not perform two consecutive moves from M_1 to M_2; and
- Scenario (b): the robot makes at least once two consecutive moves from M_1 to M_2 (and thus also at least once two consecutive moves from M_2 to M_3).

Let us first consider Scenario (a). In this scenario the robot moves each job from M_1 to M_2 and then to M_3 before moving to the next job, i.e., after moving J_j ($j = 1, ..., n$) from M_1 to M_2, the robot waits beside M_2 for the completion of J_j on this machine and then moves it to M_3. Thus, the robot starts the move of J_1 from M_1 to M_2 not earlier than at p_1, and thus starts to move J_j from M_1 to M_2 not earlier than at $p_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$ for $j = 2, ..., n$. Accordingly, J_n will not finish its processing on M_3 earlier than at

$$LB_1'(a) = p_1 + (n-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + p_2 + t_2 + p_3 =$$
$$LB_1 + (n-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i - p_1\right) \geq LB_1. \qquad (10)$$

Consider next Scenario (b), and let the *last* two consecutive moves from M_2 to M_3 be done on J_{y-1} and J_y where $2 \leq y \leq n$. Then, the earliest time in which J_{y-1} and J_y are both either in the input buffer of or processed on M_2 is at $yp_1 + t_1$, and thus the move of J_{y-1} from M_2 to M_3 will not start before this time. This implies that (i) the move of J_y from M_2 to M_3 will not start before $yp_1 + t_1 + \max\{p_2, t_2 + te_2\}$; (ii) the move of J_{y+1} from M_1 to M_2 will not start before $yp_1 + \max\{p_2, t_2 + te_2\} + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$; and that (iii) job J_y will not start its processing on M_3 before $yp_1 + \sum_{i=1}^{2} t_i + \max\{p_2, t_2 + te_2, p_3\}$. The fact that after completing the move of J_{y+1} from M_1 to M_2, the robot waits beside M_2 for the completion of each job J_j for $j = y+1, \ldots, n$ implies that the earliest time that the robot will be available to move J_n from M_1 to M_2 is

$$yp_1 + \sum_{i=1}^{2} t_i + \max\{p_2, t_2 + te_2\} + \sum_{i=1}^{2} te_i + (n-y-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2),$$

and thus we cannot start to process J_n on M_3 before

$$yp_1 + \sum_{i=1}^{2} t_i + \max\{p_2, t_2 + te_2\} + (n-y)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2).$$

To start the processing of J_n on M_3 it is also required that J_{n-1} will complete its processing on M_3. Since J_y starts its processing on M_3 not earlier than at $yp_1 + \sum_{i=1}^{2} t_i + \max\{p_2, t_2 + te_2, p_3\}$, J_{n-1} will complete its processing on M_3 not earlier than at $yp_1 + \sum_{i=1}^{2} t_i + \max\{p_2, t_2 + te_2, p_3\} + (n-y)p_3$. Thus, we have that, for any given y value, the makespan value is lower bounded by $LB_1'(b(y)) \geq LB_1$:

$$LB_1'(b(y)) = yp_1 + \sum_{i=1}^{2} t_i + p_3 +$$
$$\max\left\{\max\{p_2, t_2 + te_2\} + (n-y)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2\right), \max\{p_2, t_2 + te_2, p_3\} + (n-y)p_3\right\}. \qquad (11)$$

Let y^* be the integer y value ($y \in \{2, ..., n\}$) that minimizes $LB_1'(b(y))$ in (11). Then if Scenario (b) is selected we have that $C_{\max} \geq LB_1'(b(y^*))$, and if Scenario (a) is selected, then

$C_{\max} \geq LB_1'(a)$. Thus, a lower bound for the makespan value is given by

$$\min \left\{ LB_1'(a), LB_1'(b(y^*)) \right\}. \tag{12}$$

Consider now subcase 1.1 where in addition to the condition in (8), we also have that

$$p_1 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i. \tag{13}$$

Moreover, further divide this subcase into the following three subcases.

- Subcase 1.1.1, where $\max\{p_2, t_2 + te_2, p_3\} = p_2$. For this subcase, $LB_1'(b(y))$ in (11) can be represented by

$$LB_1'(b(y)) = yp_1 + \textstyle\sum_{i=1}^{2} t_i + p_3 + p_2 + (n - y) \left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2\right). \tag{14}$$

  The fact that the condition in (13) holds implies that (14) gets its minimum when $y = n$ (i.e., $y^* = n$). Thus, if Scenario (b) is selected then $C_{\max} \geq LB_1'(b(y^* = n)) = np_1 + \sum_{i=1}^{2} t_i + p_2 + p_3 = LB_1$, which according to (10) and (12) is also the lower bound for the makespan value in this case.

- Subcase 1.1.2, where $\max\{p_2, t_2 + te_2, p_3\} = t_2 + te_2$. For this subcase eq. (11) can be rewritten as follows:

$$LB_1'(b(y)) = yp_1 + \textstyle\sum_{i=1}^{2} t_i + p_3 + t_2 + te_2 + (n - y) \left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2\right).$$

  The fact that the condition in (13) holds implies that (14) gets its minimum when $y = n$ (i.e., $y^* = n$). Thus, if Scenario (b) is selected then

$$C_{\max} \geq LB_1'(b(y^* = n)) = np_1 + \textstyle\sum_{i=1}^{2} t_i + t_2 + te_2 + p_3 = LB_1 + t_2 + te_2 - p_2. \tag{15}$$

  According to (10), (12) and (15) a lower bound for the makespan value in this case is then given by

$$LB_1' = LB_1 + \min \left\{ (n-1) \left(\textstyle\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_1\right), t_2 + te_2 - p_2 \right\}. \tag{16}$$

- Subcase 1.1.3, where $\max\{p_2, t_2 + te_2, p_3\} = p_3$. For this subcase, according to eq. (11), we have that:

$$LB_1'(b(y)) = yp_1 + \sum_{i=1}^{2} t_i + p_3 + \max\left\{\max\{p_2, t_2 + te_2\} + (n-y)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2\right), (n-y+1)p_3\right\}. \tag{17}$$

Thus, according to (10), (12) and (17) a lower bound for the makespan value in this case is given by

$$LB_1'' = LB_1 + \min\left\{(n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_1\right), \Delta(y^*)\right\}, \tag{18}$$

where y^* is the y value ($y \in \{2, ..., n\}$) that minimizes $LB_1'(b(y))$ in (17) and $\Delta(y^*) = LB_1'(b(y^*)) - LB_1$. Note that $LB_1'(b(y))$ is a piecewise linear function of y. Thus, y^* can be easily computed in a constant time.

4.3.2 Tighter lower bounds for various subcases of Case 3

According to (5), for Case 3 where

$$p_3 = \max\left\{p_1, p_2, p_3, \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right\}, \tag{19}$$

we have that

$$LB = LB_3 = (n-1)p_3 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{3} p_i. \tag{20}$$

However, in a similar fashion to the analysis done in subsection 4.3.1, we can provide tighter lower bounds for the case where $p_3 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ and either $t_1 + te_1 = \max\{p_1, t_1 + te_1, p_2\}$ or $p_1 = \max\{p_1, t_1 + te_1, p_2\}$. For the sake of brevity, and due to similarities to what was done in subsection 4.3.1, we provide the bounds without including the formal analysis. For the first case, where $p_3 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ and $t_1 + te_1 = \max\{p_1, t_1 + te_1, p_2\}$, the tighter bound is given by

$$LB_3' = LB_3 + \min\left\{(n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3\right), (t_1 + te_1) - p_2\right\}. \tag{21}$$

In addition, for the second case, where $p_3 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ and $p_1 = \max\{p_1, t_1 + te_1, p_2\}$, the tighter bound is given by

$$LB_3'' = LB_3 + \min\left\{(n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3\right), \Delta(x^*)\right\}. \tag{22}$$

Here x^* is the integer x value ($x \in \{1, 2, ..., n-1\}$) that minimizes $LB_3'(x)$ which is given by

$$LB_3'(x) = \max\left\{(x-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2\right) + \max\{t_1 + te_1, p_2\}, xp_1\right\} \\ + p_1 + \sum_{i=1}^{2} t_i + (n-x+1)p_3, \qquad (23)$$

and $\Delta(x^*) = LB_3''(x^*) - LB_3$. Note that $LB_3'(x)$ is a piecewise linear function of x. Thus, x^* can be easily computed in a constant time.

4.3.3 Tighter lower bounds for various subcases of Case 4

For Case 4 we have that

$$\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i = \max\left\{p_1, p_2, p_3, \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right\}. \qquad (24)$$

For various subcases of this case, we revise the lower bound in eq. (4) by tighten it. The revised lower bound is given by

$$LB_r' = p_1 + n\sum_{i=1}^{2} t_i + (n-1)\sum_{i=1}^{2} te_i + p_3 + LB(I) = LB_r + LB(I), \qquad (25)$$

where $LB(I)$ is a lower bound on the robot idle time in any feasible schedule (excluding the idle times of p_1 and p_3 units of time at the beginning and at the end of the schedule).

Consider the same two possible scenarios, we consider in subsection 4.3.1 (Scenarios (a) and (b)). Similar to what is done in subsection 4.3.1, if we schedule the robot according to Scenario (a) then the makespan value is lower bounded by

$$LB_r'(a) = (n-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + \sum_{i=1}^{2} t_i + \sum_{i=1}^{3} p_i = LB_r + np_2.$$

Consider next case (b) where the robot makes two consecutive moves from M_1 to M_2 at least once. Let the *first* two consecutive moves from M_1 to M_2 be done on J_x and J_{x+1} with $1 \le x \le n-1$, and the *last* two consecutive moves from M_2 to M_3 be done on J_{y-1} and J_y with $x+1 \le y \le n$. Accordingly, the move of J_x from M_1 to M_2 will not start before $p_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$, which further implies that the move of J_{x+1} from M_1 to M_2 will not start before

$$\max\left\{p_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + te_1, (x+1)p_1\right\}.$$

Thus, the move of J_x from M_2 to M_3 cannot start before (i)

$$\max\left\{p_1+(x-1)\left(p_2+\sum_{i=1}^{2}t_i+\sum_{i=1}^{2}te_i\right)+t_1+te_1,(x+1)p_1\right\}+t_1,$$

which is the earliest time that the move of J_{x+1} from M_1 to M_2 is completed; and (ii) $p_1+(x-1)\left(p_2+\sum_{i=1}^{2}t_i+\sum_{i=1}^{2}te_i\right)+t_1+p_2$, which is the earliest time J_x is completed on M_2. Thus, the move of J_x from M_2 to M_3 cannot start before

$$T=\max\left\{p_1+(x-1)\left(p_2+\sum_{i=1}^{2}t_i+\sum_{i=1}^{2}te_i\right)+\max\{t_1+te_1,p_2\},(x+1)p_1\right\}+t_1.$$

From this time point the robot has to perform an additional $y-x-1$ non-empty moves from M_i to M_{i+1}, and $y-x-1$ empty moves from M_{i+1} to M_i for $i=1,2$ before he can start to move J_{y-1} from M_2 to M_3. Therefore, the move of J_{y-1} from M_2 to M_3 will not start before $T+(y-x-1)\left(\sum_{i=1}^{2}t_i+\sum_{i=1}^{2}te_i\right)$, and the earliest time in which J_{y-1} finishes its processing on M_3 is at $T+(y-x-1)\left(\sum_{i=1}^{2}t_i+\sum_{i=1}^{2}te_i\right)+t_2+p_3$. Moreover, the robot will be available to move J_y from M_2 to M_3 not earlier then at $T+(y-x-1)\left(\sum_{i=1}^{2}t_i+\sum_{i=1}^{2}te_i\right)+\max\{t_2+te_2,p_2\}$. The fact that the robot waits beside M_2 for the completion of each job J_j for $j=y+1,\dots,n$ further implies that the earliest time that the robot will be available for moving J_n from M_2 to M_3 is

$$T+(y-x-1)\left(\sum_{i=1}^{2}t_i+\sum_{i=1}^{2}te_i\right)+\max\{t_2+te_2,p_2\}+(n-y)\left(p_2+\sum_{i=1}^{2}t_i+\sum_{i=1}^{2}te_i\right).$$

Furthermore, the fact that J_{y-1} finishes its processing on M_3 not earlier than at $T+(y-x-1)\left(\sum_{i=1}^{2}t_i+\sum_{i=1}^{2}te_i\right)+t_2+p_3$ implies that J_{n-1} finishes its processing on M_3 not earlier then at $T+(y-x-1)\left(\sum_{i=1}^{2}t_i+\sum_{i=1}^{2}te_i\right)+t_2+(n-y+1)p_3$. Thus, a lower bound for makespan value in case (b) for a given x and y values is

$$\begin{aligned}LB_r'(b(x,y))=\max&\left\{p_1+(x-1)\left(p_2+\sum_{i=1}^{2}t_i+\sum_{i=1}^{2}te_i\right)+\max\{t_1+te_1,p_2\},(x+1)p_1\right\}\\&+t_1+(y-x-1)\left(\sum_{i=1}^{2}t_i+\sum_{i=1}^{2}te_i\right)+t_2+\\&\max\left\{\max\{t_2+te_2,p_2\}+(n-y)\left(p_2+\sum_{i=1}^{2}t_i+\sum_{i=1}^{2}te_i\right),(n-y+1)p_3\right\}+p_3.\end{aligned}\tag{26}$$

Let x^* and y^* be the integer x and y value ($x\in\{1,...,n-1\}$ and $y\in\{x+1,...,n-1\}$) that minimizes $LB_r'(b(x,y))$ in (26). Then, if Scenario (b) is selected we have that $C_{\max}\geq LB_r'(b(x^*,y^*))$, and if Scenario (a) is selected we have that $C_{\max}\geq LB_1'(a)$. Thus, a lower bound for the makespan value is given by

$$\min\left\{LB'_r(a), LB'_r(b(x^*, y^*))\right\}. \tag{27}$$

We below further divide Case 4 into subcases and provide a tighter lower bound for each of those subcases.

- Subcase 4.1, where $\max\{t_1+te_1, p_2\} \geq p_1$ and $\max\{t_2+te_2, p_2\} \geq p_3$. For this subcase, we can rewrite eq. (26) as follows:

$$\begin{aligned} LB'_r(b(x,y)) = p_1 + (x-1)\left(p_2 + \textstyle\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) + \max\{t_1+te_1, p_2\} + t_1 + \\ (y-x-1)\left(\textstyle\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) + t_2 + \max\{t_2+te_2, p_2\} + \\ (n-y)\left(p_2 + \textstyle\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) + p_3. \end{aligned} \tag{28}$$

One can easily observe from (28) that $x^* = 1$and $y^* = n$. The fact that

$$\begin{aligned} LB'_r(b(x^*=1, y^*=n)) = p_1 + \max\{t_1+te_1, p_2\} + t_1 + \\ (n-2)\left(\textstyle\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) + t_2 + \max\{t_2+te_2, p_2\} + p_3 = \\ LB_r + \max\{t_1+te_1, p_2\} + \max\{t_2+te_2, p_2\} - \left(\textstyle\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right), \end{aligned}$$

and that

$$\max\{t_1+te_1, p_2\} + \max\{t_2+te_2, p_2\} - \left(\textstyle\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) \leq np_2,$$

implies that $LB'_r(b(x^*, y^*)) \leq LB'_r(a)$. Thus, according to (27) a lower bound for the makespan value is given by

$$LB'_r = LB_r + \max\{t_1+te_1, p_2\} + \max\{t_2+te_2, p_2\} - \left(\textstyle\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right). \tag{29}$$

- Subcase 4.2, where $\max\{t_1+te_1, p_2\} \geq p_1$ and $\max\{t_2+te_2, p_2\} < p_3$. For this subcase, the value of $LB'_r(b(x,y))$ in eq. (26) becomes:

$$\begin{aligned} LB'_r(b(x,y)) = p_1 + (x-1)\left(p_2 + \textstyle\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) + \max\{t_1+te_1, p_2\} + t_1 + \\ (y-x-1)\left(\textstyle\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) + t_2 + \\ \max\left\{\max\{t_2+te_2, p_2\} + (n-y)\left(p_2 + \textstyle\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right), (n-y+1)p_3\right\} + p_3. \end{aligned} \tag{30}$$

One can easily observe from (30) that $x^* = 1$. Thus, if Scenario (b) is selected then

$$C_{\max} \geq LB'_r(b(x^* = 1, y)) = p_1 + \max\{t_1 + te_1, p_2\} + t_1 + (y-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_2 + \max\left\{\max\{t_2 + te_2, p_2\} + (n-y)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), (n-y+1)p_3\right\} + p_3. \quad (31)$$

It follows from (27) that a lower bound for the makespan value is given by

$$LB''_r = LB_r + \min\{np_2, \Delta(y^*)\}, \quad (32)$$

where y^* is the y value ($y \in \{2, ..., n\}$) that minimizes $LB'_r(b(x^* = 1, y))$ in (31) and $\Delta(y^*) = LB'_r(b(x^* = 1, y^*)) - LB_r$. Note that $LB'_r(b(x^* = 1, y))$ in (31) is a piecewise linear function of y. Thus, y^* can be easily computed in a constant time.

- Subcase 4.3, where $\max\{t_1 + te_1, p_2\} < p_1$ and $\max\{t_2 + te_2, p_2\} \geq p_3$. For this subcase, eq. (26) reduces to

$$LB'_r(b(x, y)) = \max\left\{p_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + \max\{t_1 + te_1, p_2\}, (x+1)p_1\right\} + t_1 + (y-x-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_2 + \max\{t_2 + te_2, p_2\} + (n-y)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + p_3. \quad (33)$$

One can easily observe from (33) that $y^* = n$. Thus, if Scenario (b) is selected then

$$C_{\max} \geq LB'_r(b(x, y^* = n)) = \max\left\{p_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + \max\{t_1 + te_1, p_2\}, (x+1)p_1\right\} + t_1 + (n-x-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + \max\{t_2 + te_2, p_2\} + t_2 + p_3. \quad (34)$$

It follows from (27) that a lower bound for the makespan value is given by

$$LB'''_r = LB_r + \min\{np_2, \Delta(x^*)\}, \quad (35)$$

where x^* is the x value ($x \in \{1, ..., n-2\}$) that minimizes $LB'_r(b(x, y^* = n))$ in (34) and $\Delta(x^*) = LB'_r(b(x^*, y^* = n)) - LB_r$. Note that $LB'_r(b(x, y^* = n))$ in (34) is a piecewise linear function of x. Thus, x^* can be easily computed in a constant time.

- Subcase 4.3, where $\max\{t_1 + te_1, p_2\} < p_1$ and $\max\{t_2 + te_2, p_2\} < p_3$. According to

(27), a lower bound for the makespan value is given by

$$LB_r'''' = \min\{LB_r'(a), LB_r'(b(x^*, y^*))\} = LB_r + \min\{np_2, \Delta(x^*, y^*)\},$$

where x^* and y^* are the integer x and y value ($x \in \{1, ..., n-1\}$ and $y \in \{x+1, ..., n\}$) that minimizes $LB_r'(b(x, y))$ in (26) and $\Delta(x^*, y^*) = LB_r'(b(x^*, y^*) - LB_r$. Note that $LB_r'(b(x, y))$ is a piecewise linear function of x and y. Thus, here as well, x^* and y^* can be easily computed in a constant time.

4.4 Optimal schedules

Below, we provide an optimization algorithm for solving the $F3, R1|p_{ij} = p_i, t_{ij} = t_i|C_{\max}$ problem. This is done by decomposing the problem into a set of sub-problems (cases), and by providing an optimal schedule for each sub-problem, separately. This is followed by a formal proof that the provided schedules are indeed optimal for all subcases of subcase 1.1.3. Due to similarity and for the sake of brevity, we include the analysis of complementary subcases of *Case 1* (subcases 1.1.1, 1.1.2 and 1.2) and *Cases 2, 3,* and *4* in appendices (Sections 9.1, 9.2, 9.3, and 9.4, respectively).

Algorithm 2 *Optimal schedules for the* $F3, R1|p_{ij} = p_i, t_{ij} = t_i|C_{\max}$ *problem*

```
Input: $m, n, p_i$ for $i = 1, 2, 3$, $t_i$ and $te_i$ for $i = 1, 2$.
If (Case 1) $p_1 = \max\{p_1, p_2, p_3, \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\}$ then goto Procedure Case 1
If (Case 2) $p_2 = \max\{p_1, p_2, p_3, \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\}$ then Sequence 2 is optimal and
    $C_{\max} = LB_2$.
If (Case 3) $p_3 = \max\{p_1, p_2, p_3, \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\}$ then goto Procedure Case 3
If (Case 4) $\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i = \max\{p_1, p_2, p_3, \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\}$ then goto Procedure
    Case 4

Procedure Case 1
If (subcase 1.1) $p_1 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ then
    If (subcase 1.1.1) $p_2 = \max\{p_2, p_3, t_2 + te_2\}$
        Sequence 2 with $x = 1$ and $y = n$ is optimal and $C_{\max} = LB_1$.
    If (subcase 1.1.2) $t_2 + te_2 = \max\{p_2, p_3, t_2 + te_2\}$
        If (subcase 1.1.2.1) $t_2 + te_2 - p_2 \leq (n-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i - p_1\right)$
            Sequence 2 with $x = 1$ and $y = n$ is optimal and $C_{\max} = LB_1'$.
        If (subcase 1.1.2.2) $t_2 + te_2 - p_2 > (n-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i - p_1\right)$
            Sequence 1 is optimal and $C_{\max} = LB_1'$.
    If (subcase 1.1.3) $p_3 = \max\{p_2, p_3, t_2 + te_2\}$
        Let $y^*$ be the $y$ value ($y \in \{2, ..., n\}$) that minimizes $LB_1'(b(y))$ given by (17)
            and let $\Delta(y^*) = LB_1'(b(y^*)) - LB_1$.
        If (subcase 1.1.3.1) $(n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_1\right) \leq \Delta(y^*)$
            Sequence 1 is optimal and $C_{\max} = LB_1''$.
        If (subcase 1.1.3.2) $(n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_1\right) > \Delta(y^*)$
            Sequence 2 with $x^* = 1$ and $y = y^*$ is optimal and $C_{\max} = LB_1''$.
If (subcase 1.2) $p_1 \geq p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ then
    Sequence 1 is optimal and $C_{\max} = LB_1$.
```

Procedure Case 3

```
If (subcase 3.1) $p_3 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$
    If (subcase 3.1.1) $p_2 = \max\{p_1, t_1 + te_1, p_2\}$
        Sequence 2 with $x = 1$ and $y = n$ is optimal and $C_{\max} = LB_3$.
    If (subcase 3.1.2) $p_1 = \max\{p_1, t_1 + te_1, p_2\}$
        Let $x^*$ be the $x$ value ($x \in \{1, ..., n-1\}$) that minimizes $LB_3'(x)$ given by (23)
            and let $\Delta(x^*) = LB_3'(x^*) - LB_3$.
        If (subcase 3.1.2.1) $(n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3) \leq \Delta(x^*)$
            Sequence 1 is optimal and $C_{\max} = LB_3''$.
        If (subcase 3.1.2.2) $(n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3) > \Delta(x^*)$
            Sequence 2 with $x = x^*$ and $y = n$ is optimal and $C_{\max} = LB_3''$.
    If (subcase 3.1.3) $t_1 + te_1 = \max\{p_1, t_1 + te_1, p_2\}$
        If (subcase 3.1.3.1) $t_1 + te_1 - p_2 \leq (n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3)$
            Sequence 2 with $x = 1$ and $y = n$ is optimal and $C_{\max} = LB_3'$.
        If (subcase 3.1.3.2) $t_1 + te_1 - p_2 > (n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3)$
            Sequence 1 is optimal and $C_{\max} = LB_3'$.
If (subcase 3.2) $p_3 \geq p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$
    Sequence 1 is optimal and $C_{\max} = LB_3$.
```

Procedure Case 4

```
If (subcase 4.1) $\max\{t_1 + te_1, p_2\} \geq p_1$ and $\max\{t_2 + te_2, p_2\} \geq p_3$
    Sequence 2 with $x = 1$ and $y = n$ is optimal and $C_{\max} = LB_r'$.
If (subcase 4.2) $\max\{t_1 + te_1, p_2\} \geq p_1$ and $\max\{t_2 + te_2, p_2\} < p_3$
    Let $y^*$ be the $y$ value ($y \in \{2, ..., n\}$) that minimizes $LB_r'(b(x^* = 1, y))$ given by (31),
        and let $\Delta(y^*) = LB_r'(b(x^* = 1, y^*)) - LB_r$.
        If (subcase 4.2.1) $np_2 \leq \Delta(y^*)$
            Sequence 1 is optimal and $C_{\max} = LB_r''$.
        If (subcase 4.2.2) $np_2 > \Delta(y^*)$
            Sequence 2 with $x = 1$ and $y = y^*$ is optimal and $C_{\max} = LB''_r$.
    If (subcase 4.3) $\max\{t_1 + te_1, p_2\} < p_1$ and $\max\{t_2 + te_2, p_2\} \geq p_3$
        Let $x^*$ be the $x$ value ($x \in \{1, ..., n-1\}$) that minimizes $LB'_r(b(x, y^* = n))$ given by
        (34), and let $\Delta(x^*) = LB'_r(b(x^*, y^* = n)) - LB_r$.
        If (subcase 4.3.1) $np_2 \leq \Delta(x^*)$
            Sequence 1 is optimal and $C_{\max} = LB'''_r$.
        If (subcase 4.3.2) $np_2 > \Delta(x^*)$
            Sequence 2 with $x = x^*$ and $y = n$ is optimal and $C_{\max} = LB'''_r$.
    If (subcase 4.4) $\max\{t_1 + te_1, p_2\} < p_1$ and $\max\{t_2 + te_2, p_2\} < p_3$
        Let $x^*$ and $y^*$ be the $x$ and $y$ values ($x \in \{1, ..., n-1\}$ and $y \in \{x+1, ..., n\}$) that
        minimize $LB'_r(b(x, y))$ given by (26), and let $\Delta(x^*, y^*) = LB'_r(b(x^*, y^*)) - LB_r$.
        If (subcase 4.4.1) $np_2 \leq \Delta(x^*, y^*)$
            Sequence 1 is optimal and $C_{\max} = LB''''_r$.
        If (subcase 4.4.2) $np_2 > \Delta(x^*, y^*)$
            Sequence 2 with $x = x^*$ and $y = y^*$ is optimal and $C_{\max} = LB''''_r$.
```

Remark 1 *The fact that Sequences 1 and 2 provide schedules which store no more than a single job in all input and output buffers implies that our optimization algorithm can be used to solve the* $F3, R1|p_{ij} = p_i, t_{ij} = t_i, C(IB_i) = C(OB_i) = 1|C_{\max}$ *problem as well.*

In the following, for various cases of subcase 1.1.3, Algorithm 1 is used to define the schedule for the suggested sequence of robot moves. We then prove the optimality of the schedule by showing that (i) the schedule is indeed *feasible*; and that (ii) the schedule provides a makespan value which is equal to the lower bound value. We note that a schedule is feasible if it satisfies the following three conditions:

Condition 1 *There is no overlap between processing time intervals of different jobs on the same machine;*

Condition 2 *There is no overlap between different operations of the same job; and*

Condition 3 *There is no overlap between robot operations.*

4.4.1 The analysis of subcase 1.1.3

In subcase 1.1.3 $p_1 = \max\left\{p_1, p_2, p_3, \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right\}$; $p_1 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$; and $p_3 = \max\{p_2, t_2 + te_2, p_3\}$. We now prove that Algorithm 2 provides the optimal robot sequence of moves for two subcases that can arise. The first (subcase 1.1.3.1) is when $(n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_1) \leq \Delta(y^*)$, while the second (subcase 1.1.3.2) is opposite.

Optimal schedule for subcase 1.1.3.1 For subcase 1.1.3.1 we have that $(n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_1) \leq \Delta(y^*)$, and according to Algorithm 5, *Sequence 1* is optimal. By applying Algorithm 1, we construct the following schedule (Schedule 1.1.3.1) that corresponds to this sequence of moves (Schedule 1.1.3.1 is depicted in Figure 4 below for $n = 4$ jobs when empty return moves are shown in bold):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $(S_{1j}^m, C_{1j}^m] = ((j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_j on M_2 during time interval $(S_{2j}^m, C_{2j}^m] = (p_1 + t_1 + (j-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2), S_{2j}^m + p_2]$ for $j = 1, ..., n$.
- Schedule job J_j on M_3 during time interval $(S_{3j}^m, C_{3j}^m] = (\sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (j-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2), S_{3j}^m + p_3]$ for $j = 1, ..., n$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $(S_{1j}^r, C_{1j}^r] = (p_1 + (j-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2), S_{1j}^r + t_1]$ for $j = 1, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $(S_{2j}^r, C_{2j}^r] = (p_1 + t_1 + p_2 + (j-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2), S_{2j}^r + t_2]$ for $j = 1, ..., n$.
- Move the robot empty from M_3 to M_1 during time interval $(S_{3j}^m, S_{3j}^m + \sum_{i=1}^{2} te_i]$ for $j = 1, ..., n-1$.

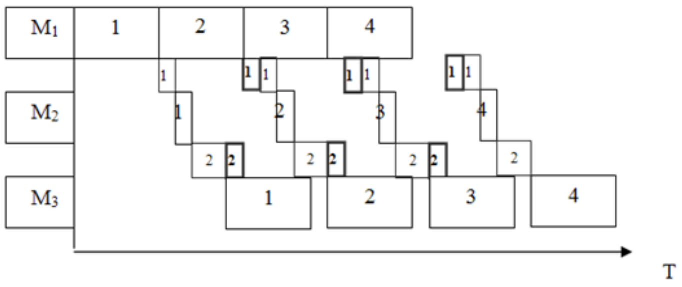

Figure 4: Schedule 1.1.3.1.

Lemma 2 *Schedule 1.1.3.1 is an optimal schedule for subcase 1.1.3.1.*

Proof. We start by showing that Schedule 1.1.3.1 is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$; that

$$S^m_{2,j+1} = p_1 + t_1 + j(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) > C^m_{2j} = p_1 + t_1 + (j-1)\,(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) + p_2;$$

and that

$$\begin{aligned} S^m_{3,j+1} &= \sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + j(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) > \\ C^m_{3j} &= \sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (j-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) + p_3 \end{aligned}$$

for $j = 1, ..., n-1$, where the last inequality follows from the fact that $p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i > p_1 \geq p_3$ implies that there is no overlap between the job processing in any machine. Thus, Condition 1 holds. Moreover, the fact that $C^m_{1j} = jp_1 < S^r_{1j} = p_1 + (j-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2)$; that $C^r_{1j} = S^m_{2j} = p_1 + (j-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) + t_1$; that $C^m_{2j} = S^r_{2j} = p_1 + t_1 + (j-1)\,(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) + p_2$; and that $C^r_{2j} = S^m_{3j} = p_1 + t_1 + (j-1)\,(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) + p_2 + t_2$, implies that there is no overlap between the processing and transferring operations of job J_j for $j = 1, \ldots n$. Thus, Condition 2 holds.

Since

$$C^r_{1j} = p_1 + (j-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) + t_1 < S^r_{2j} = p_1 + t_1 + (j-1)\,(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) + p_2$$

there is no overlap between the move of J_j from M_1 to M_2 and the move of J_j from M_2 to M_3. Moreover, after completing the move of J_j from M_2 to M_3, the robot returns to M_1 at $\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} p_i + (j-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) + \sum_{i=1}^{2} te_i = S^r_{1,j+1}$ for $j = 1, ..., n-1$, which implies that there is no overlap between the moves of J_j from M_2 to M_3 and of J_{j+1} from M_1 to M_2. Therefore, there is no overlap between the robot operations and Condition 3 holds.

The feasibility of Schedule 1.1.3.1 implies that the completion time of J_n on machine M_3 is at

$$\begin{aligned} C^m_{3n} &= \sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (n-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + p_3 = \\ &LB_1 + (n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_1\right). \end{aligned}$$

Furthermore, the fact that this time matches the lower bound in eq. (18) when $(n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_1) \le \Delta(y^*)$ implies that this schedule is optimal for subcase 1.1.3.1. ■

Optimal schedules for subcase 1.1.3.2 For subcase 1.1.3.2 we have that $(n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_1\right) > \Delta(y^*)$. Consider first the case where $y^* = n$. According to Algorithm 2, *Sequence 2* with $x = 1$ and $y = n$ is optimal. By applying Algorithm 1, we construct the following schedule (Schedule 1.1.3.2(a)) that corresponds to this sequence of moves (Schedule 1.1.3.2(a) is depicted in Figure 5 below for $n = 4$ jobs with empty return moves shown in bold):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $(S^m_{1j}, C^m_{1j}] = ((j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_j on M_2 during time interval $(S^m_{2j}, C^m_{2j}] = (jp_1+t_1, S^m_{2j}+p_2]$ for $j = 1, ..., n$.
- Schedule job J_j on M_3 during time interval $(S^m_{3j}, C^m_{3j}] = ((j+1)p_1 + \sum_{i=1}^{2} t_i, S^m_{3j} + p_3]$ for $j = 1, ..., n-1$.
- Schedule job J_n on M_3 during time interval $(S^m_{3n}, C^m_{3n}] = (np_1 + \sum_{i=1}^{2} t_i + p_3, S^m_{3n} + p_3]$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $(S^r_{1j}, C^r_{1j}] = (jp_1, S^r_{1j} + t_1]$ for $j = 1, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $(S^r_{2j}, C^r_{2j}] = ((j+1)p_1 + t_1, S^r_{2j} + t_2]$ for $j = 1, ..., n-1$.
- Move job J_n from M_2 to M_3 during time interval $(S^r_{2n}, C^r_{2n}] = (np_1 + t_1 + \max\{p_2, t_2 + te_2\}, S^r_{2n} + t_2]$.
- Move the robot empty from M_2 to M_1 during time interval $(p_1 + t_1, p_1 + t_1 + te_1]$.
- Move the robot empty from M_3 to M_1 during time interval $((j+1)p_1 + \sum_{i=1}^{2} t_i, (j+1)p_1 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i]$ for $j = 1, ..., n-2$.
- Move the robot empty from M_3 to M_2 during time interval $(np_1+\sum_{i=1}^{2} t_i, np_1+\sum_{i=1}^{2} t_i+te_2]$.

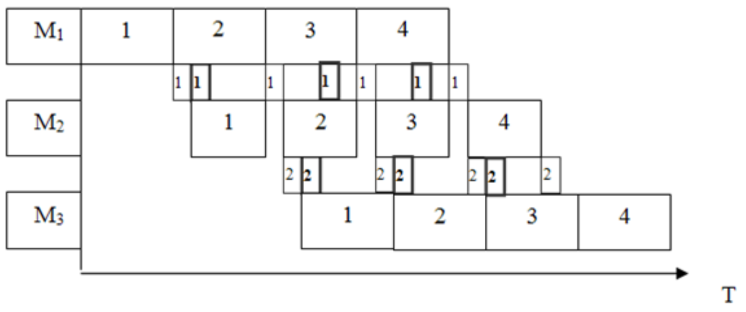

Figure 5: Schedule 1.1.3.2(a)

Lemma 3 *Schedule 1.1.3.2(a) is an optimal schedule for subcase 1.1.3.2 when* $y^* = n$.

Proof. We start by showing that Schedule 1.1.3.2(a) is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$ for $j = 1, ..., n-1$; that $S^m_{2,j+1} = (j+1)p_1 + t_1 \geq C^m_{2j} = jp_1 + t_1 + p_2$ for $j = 1, ..., n-1$ (since $p_2 \leq p_1$); that $S^m_{3,j+1} = (j+2)p_1 + \sum_{i=1}^{2} t_i \geq C^m_{3j} = (j+1)p_1 + \sum_{i=1}^{2} t_i + p_3$ for $j = 1, ..., n-2$ (since $p_3 \leq p_1$); and that $S^m_{3n} = np_1 + \sum_{i=1}^{2} t_i + p_3 = C^m_{3,n-1}$, implies that there are no overlap between processing of jobs in any machine. Thus, Condition 1 holds. Moreover, the fact that $C^m_{1j} = S^r_{1j} = jp_1$; that $C^r_{1j} = jp_1 + t_1 = S^m_{2j}$; that $C^m_{2j} = jp_1 + t_1 + p_2 \leq S^r_{2j} = (j+1)p_1 + t_1$ for $j = 1, ..., n-1$ (since $p_2 \leq p_1$); that $C^m_{2n} = np_1 + t_1 + p_2 \leq S^r_{2n} = np_1 + t_1 + \max\{p_2, t_2 + te_2\}$; that $C^r_{2j} = (j+1)p_1 + t_1 + t_2 = S^m_{3j}$ for $j = 1, ..., n-1$; and that $C^r_{2n} = np_1 + \sum_{i=1}^{2} t_i + \max\{p_2, t_2 + te_2\} \leq S^m_{3n} = np_1 + \sum_{i=1}^{2} t_i + p_3$ (since $p_3 = \max\{p_2, t_2 + te_2, p_3\}$) implies that there is no overlap between transferring operations of job J_j for $j = 1, \ldots n$. Thus, Condition 2 holds.

The robot starts its moves by moving J_1 from M_1 to M_2 during time interval $(p_1, p_1 + t_1]$. Then the robot returns empty to M_1 during time interval $(p_1 + t_1, p_1 + t_1 + te_1]$, and moves J_2 from M_1 to M_2 during time interval $(2p_1, 2p_1 + t_1]$. Since $2p_1 \geq p_1 + t_1 + te_1$, there is no overlap between these two last operations. Then, for $j = 1, ..., n-2$, the robot moves J_j from M_2 to M_3 during time interval $((j+1)p_1 + t_1, (j+1)p_1 + \sum_{i=1}^{2} t_i]$; returns empty to M_1 during time interval $((j+1)p_1 + \sum_{i=1}^{2} t_i, (j+1)p_1 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i]$; and moves J_{j+2} from M_1 to M_2 during time interval $((j+2)p_1, (j+2)p_1 + t_1]$. The fact that $(j+2)p_1 \geq (j+1)p_1 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ implies that there is no overlap between these operations as well. Lastly, after moving J_n from M_1 to M_2 during time interval $(S^r_{1n}, C^r_{1n}] = (np_1, np_1 + t_1]$, the robot moves J_{n-1} from M_2 to M_3 during time interval $(np_1 + t_1, np_1 + t_1 + t_2]$; returns empty to M_2 during time interval $(np_1 + \sum_{i=1}^{2} t_i, np_1 + \sum_{i=1}^{2} t_i + te_2]$; and finishes its operations by moving J_n from

M_2 to M_3 during time interval $(np_1+t_1+\max\{p_2, t_2+te_2\}, np_1+\sum_{i=1}^{2} t_i+\max\{p_2, t_2+te_2\}]$. Therefore, there is no overlap between the robot operations and Condition 3 holds.

The feasibility of Schedule 1.1.3.2(a) implies that the completion time of job J_n on machine M_3 is at $C_{3n}^m = np_1 + \sum_{i=1}^{2} t_i + 2p_3$. The fact that this time matches the lower bound in eq. (18) when $(n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_1\right) > \Delta(y^*)$ and $y^* = n$ implies that this schedule is optimal for subcase 1.2.3.2.

■

Consider next the case where $y^* < n$. According to Algorithm 2, *Sequence 2* with $x = 1$ and $y = y^*$ is optimal. By applying Algorithm 1, we construct the following schedule (Schedule 1.1.3.2(b)) that corresponds to this sequence of moves (Schedule 1.1.3.2(b) is depicted in Figure 6 below for $n = 5$ jobs and $y = 2$ with empty return moves shown in bold):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $(S_{1j}^m, C_{1j}^m] = ((j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_j on M_2 during time interval $(S_{2j}^m, C_{2j}^m] = (jp_1+t_1, S_{2j}^m+p_2]$ for $j = 1, ..., y$.
- Schedule job J_j on M_2 during time interval $(S_{2j}^m, C_{2j}^m] = (yp_1 + t_1 + \max\{p_2, te_2 + t_2\} + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + (j-y-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2), S_{2j}^m + p_2]$ for $j = y+1, ..., n$.
- Schedule job J_j on M_3 during time interval $(S_{3j}^m, C_{3j}^m] = ((j+1)p_1 + t_1 + t_2, S_{3j}^m + p_3]$ for $j = 1, ..., y-1$.
- Schedule job J_y on M_3 during time interval $(S_{3y}^m, C_{3y}^m] = (yp_1 + t_1 + t_2 + p_3, S_{3y}^m + p_3]$.
- Schedule job J_j on M_3 during time interval $(S_{3j}^m, C_{3j}^m] = (\max\{yp_1 + t_1 + t_2 + (j-y+1)p_3, C_{2j}^m + t_2\}, S_{3j}^m + p_3]$ for $j = y+1, ..., n$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $(S_{1j}^r, C_{1j}^r] = (jp_1, S_{1j}^r + t_1]$ for $j = 1, ..., y$.
- Move job J_{y+1} from M_1 to M_2 during time interval $(S_{1,y+1}^r, C_{1,y+1}^r] = (yp_1 + \sum_{i=1}^{2} t_i + \max\{p_2, te_2 + t_2\} + \sum_{i=1}^{2} te_i, S_{1,y+1}^r + t_1]$.
- Move job J_j from M_1 to M_2 during time interval $(S_{1j}^r, C_{1j}^r] = (S_{1,y+1}^r + (j-y-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2), S_{1j}^r + t_1]$ for $j = y+2, ..., n$.

- Move job J_j from M_2 to M_3 during time interval $(S^r_{2j}, C^r_{2j}] = ((j+1)p_1 + t_1, S^r_{2j} + t_2]$ for $j = 1, ..., y-1$.
- Move job J_y from M_2 to M_3 during time interval $(S^r_{2y}, C^r_{2y}] = (yp_1 + t_1 + \max\{p_2, te_2 + t_2\}, S^r_{2y} + t_2]$.
- Move job J_j from M_2 to M_3 during time interval $(S^r_{2j}, C^r_{2j}] = (C^m_{2j}, S^r_{2j} + t_2]$ for $j = y+1, ..., n$.
- Move the empty robot from M_2 to M_1 during time interval $(p_1 + t_1, p_1 + t_1 + te_1]$.
- Move the robot empty from M_3 to M_1 during time interval $((j+1)p_1 + \sum_{i=1}^{2} t_i, (j+1)p_1 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i]$ for $j = 1, ..., y-2$.
- Move the robot empty from M_3 to M_2 during time interval $(yp_1 + \sum_{i=1}^{2} t_i, yp_1 + \sum_{i=1}^{2} t_i + te_2]$.
- Move the robot empty from M_3 to M_1 during time interval $(C^r_{2j}, C^r_{2j} + \sum_{i=1}^{2} te_i]$ for $j = y, ..., n-1$.

We below prove that for any given $y \in \{2, ..., n-1\}$ value, Schedule 1.1.3.2(b) is a feasible schedule with a makespan value of $LB_1'' = LB_1 + \Delta(y)$. The fact that this makespan value matches the lower bound value in (18) when $y = y^*$, implies that Schedule 1.1.3.2(b) is an optimal schedule when $y = y^*$.

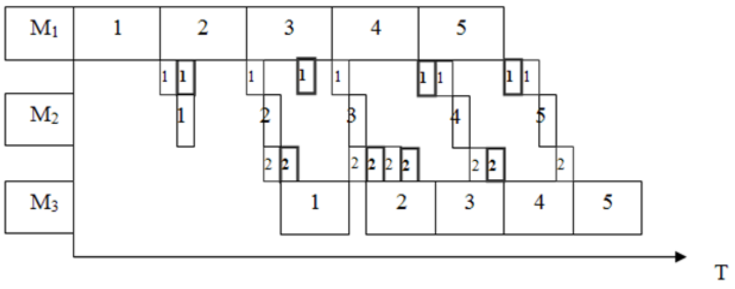

Figure 6: Schedule 1.1.3.2(b).

Lemma 4 *Schedule 1.1.3.2(b) is an optimal schedule for subcase 1.1.3.2 when* $y^* < n$.

Proof. We start by showing that Schedule 1.1.3.2(b) is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$ for $j = 1, ..., n-1$ implies that there is no overlap between the processing of any two consecutive jobs on M_1. Moreover, the fact $S^m_{2,j+1} = (j+1)p_1 + t_1 \geq C^m_{2j} = jp_1 + t_1 + p_2$ for $j = 1, ..., y-1$ (since $p_1 \geq p_2$); that

$$S^m_{2,y+1} = yp_1 + t_1 + \max\{p_2, te_2 + t_2\} + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i > C^m_{2y} = yp_1 + t_1 + p_2;$$

that $S^m_{2,y+2} = S^m_{2,y+1} + (\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2) > C^m_{2,y+1} = S^m_{2,y+1} + p_2$; and that

$$S^m_{2,j+1} = S^m_{2,y+1} + (j-y)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) > C^m_{2j} = S^m_{2,y+1} + (j-y-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) + p_2$$

for $j = y+2, ..., n-1$ implies that there is no overlap between the processing of any two consecutive jobs on M_2. In addition, the fact that $S^m_{3,j+1} = (j+2)p_1 + t_1 + t_2 \geq C^m_{3j} = (j+1)p_1 + t_1 + t_2 + p_3$ for $j = 1, ..., y-2$ (since $p_1 \geq p_3$); that $S^m_{3y} = yp_1 + t_1 + t_2 + p_3 = C^m_{3,y-1}$; and that

$$S^m_{3,y+1} = \max\{yp_1 + t_1 + t_2 + 2p_3, C^m_{2,y+1} + t_2\} \geq C^m_{3y} = yp_1 + t_1 + t_2 + 2p_3$$

implies that in order to prove that there is no overlap between the processing of any two consecutive jobs on M_3, it only remains to prove that

$$\begin{aligned} S^m_{3,j+1} = \max\{yp_1 + t_1 + t_2 + (j-y+2)p_3, C^m_{2,j+1} + t_2\} \geq \\ C^m_{3j} = \max\{yp_1 + t_1 + t_2 + (j-y+1)p_3, C^m_{2j} + t_2\} + p_3 \end{aligned} \qquad (36)$$

for $j = y+1, ..., n-1$. It is easy to observe that the inequality in (36) holds if

$$C^m_{2,j+1} \geq C^m_{2j} + p_3 \qquad (37)$$

for $j = y+1, ..., n-1$. For $j = y+1$, the inequality in (37) reduces to

$$C^m_{2,y+2} = S^m_{2,y+1} + \left(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2\right) + p_2 \geq C^m_{2,y+1} + p_3. \qquad (38)$$

The fact that the inequality in (38) holds follows from the fact that $C^m_{2,y+1} = S^m_{2,y+1} + p_2$ and that $\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2 > p_1 \geq p_3$, which implies that the condition in (37) holds when

$j = y + 1$. For $j = y + 2, ..., n - 1$, the condition in (37) reduces to

$$S^m_{2,y+1} + (j - y)\left(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2\right) + p_2 \geq$$
$$S^m_{2,y+1} + (j - y - 1)\left(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2\right) + p_2 + p_3. \tag{39}$$

The fact that the inequality in (39) holds follows from the fact that $\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2 > p_1 \geq p_3$, and thus the condition in (37) holds also when $j = y + 2, ..., n - 1$. This completes our proof that there is no overlap between the processing of any two consecutive jobs on any machine, and thus Condition 1 holds.

The fact that $C^m_{1j} = jp_1 = S^r_{1j}$ for $j = 1, ..., y$; that

$$C^m_{1,y+1} = (y + 1)p_1 < S^r_{1,y+1} = yp_1 + \sum_{i=1}^2 t_i + \max\{p_2, te_2 + t_2\} + \sum_{i=1}^2 te_i;$$

and that

$$\begin{aligned} C^m_{1j} &= jp_1 < S^r_{1j} = S^r_{1,y+1} + (j - y - 1)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2) = \\ & yp_1 + \sum_{i=1}^2 t_i + \max\{p_2, te_2 + t_2\} + \sum_{i=1}^2 te_i + (j - y - 1)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2) \end{aligned}$$

for $j = y + 2, ..., n$, where the two last inequalities follow from the fact that $p_1 < p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i$ implies that there is no overlap between the processing of J_j on M_1 and the move of J_j from M_1 to M_2 for $j = 1, ..., n$. Moreover, the fact that $C^r_{1j} = jp_1 + t_1 = S^m_{2j}$ for $j = 1, ..., y$; that

$$C^r_{1,y+1} = yp_1 + \sum_{i=1}^2 t_i + \max\{p_2, te_2 + t_2\} + \sum_{i=1}^2 te_i + t_1 = S^m_{2,y+1};$$

and that

$$C^r_{1j} = S^r_{1,y+1} + t_1 + (j - y - 1)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2) = S^m_{2j} = S^m_{2,y+1} + (j - y - 1)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2)$$

for $j = y + 2, ..., n$ implies that there is no overlap between the move of J_j from M_1 to M_2 and the processing of J_j on M_2 for $j = 1, ..., n$. In addition, the fact that $C^m_{2j} = jp_1 + t_1 + p_2 \leq S^r_{2j} = (j + 1)p_1 + t_1$ for $j = 1, ..., y - 1$ (since $p_1 \geq p_2$); that $C^m_{2y} = yp_1 + t_1 + p_2 \leq S^r_{2y} = yp_1 + t_1 + \max\{p_2, te_2 + t_2\}$; and that $C^m_{2j} = S^r_{2j}$ for $j = y + 1, ..., n$, implies that there is no overlap between the processing of job J_j on M_2 and the move of J_j from M_2 to M_3 for $j = 1, ..., n$. Lastly, the fact that $C^r_{2j} = (j+1)p_1 + t_1 + t_2 = S^m_{3j}$ for $j = 1, ..., y-1$; that

$$C^r_{2y} = yp_1 + t_1 + \max\{p_2, te_2 \ + t_2\} + t_2 \leq S^m_{3y} = yp_1 + t_1 + t_2 + p_3$$

(since $p_3 = \max\{p_2, t_2 + te_2, p_3\}$); and that $C^r_{2j} = C^m_{2j} + t_2 \leq S^m_{3j} = \max\{yp_1 + t_1 + t_2 + (j - y + 1)p_3, C^m_{2j} + t_2\}$ for $j = y+1, ..., n$ implies that there is no overlap between the move of J_j from M_2 to M_3 and the processing of J_j on M_3 for $j = 1, ..., n$. Thus, Condition 2 holds.

The robot starts its moves by moving J_1 from M_1 to M_2 during time interval $(p_1, p_1 + t_1]$. Then, the robot returns empty to M_1 during time interval $(p_1 + t_1, p_1 + t_1 + te_1]$, and moves J_2 from M_1 to M_2 during time interval $(2p_1, 2p_1 + t_1]$. The fact that $p_1 > t_1 + te_1$ implies that there is no overlap between these last two operations. Then, for $j = 1, ..., y-2$, the robot moves J_j from M_2 to M_3 during time interval $((j+1)p_1 + t_1, (j+1)p_1 + \sum_{i=1}^{2} t_i]$; returns empty to M_1 during time interval $((j+1)p_1 + \sum_{i=1}^{2} t_i, (j+1)p_1 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i]$; and moves J_{j+2} from M_1 to M_2 during time interval $((j+2)p_1, (j+2)p_1 + t_1]$. The fact that $(j+2)p_1 \geq (j+1)p_1 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ implies that there is no overlap between these operations as well. After completing the move of J_y from M_1 to M_2 at $yp_1 + t_1$, the robot moves J_{y-1} from M_2 to M_3 during time interval $(yp_1 + t_1, yp_1 + t_1 + t_2]$. Then the robot returns empty to M_2 during time interval $(yp_1 + t_1 + t_2, yp_1 + t_1 + t_2 + te_2]$; moves J_y from M_2 to M_3 during time interval $(yp_1 + t_1 + \max\{p_2, te_2 \ + t_2\}, yp_1 + t_1 + \max\{p_2, te_2 \ + t_2\} + t_2]$; and returns empty to M_1 during time interval

$$(yp_1 + t_1 + \max\{p_2, te_2 \ + t_2\} + t_2, yp_1 + t_1 + \max\{p_2, te_2 \ + t_2\} + t_2 + \sum_{i=1}^{2} te_i].$$

This follows by moving J_{y+1} from M_1 to M_2 during time interval

$$(yp_1 + \sum_{i=1}^{2} t_i + \max\{p_2, te_2 + t_2\} + \sum_{i=1}^{2} te_i, yp_1 + \sum_{i=1}^{2} t_i + \max\{p_2, te_2 + t_2\} + \sum_{i=1}^{2} te_i + t_1].$$

Then, for $j = y+1, ..., n-1$, the robot moves J_j from M_2 to M_3 during time interval

$$(S^r_{2j}, C^r_{2j}] = (yp_1 + t_1 + \max\{p_2, te_2 \ + t_2\} + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + (j-y-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2), S^r_{2j} + t_2];$$

returns empty to M_1 during time interval $(C^r_{2j}, C^r_{2j} + \sum_{i=1}^{2} te_i]$; and moves J_{j+1} from M_1 to

M_2 during time interval

$$(S^r_{1,j+1}, C^r_{1,j+1}] = (S^r_{1,y+1} + (j-y)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2), S^r_{1,j+1} + t_1].$$

The fact that

$$C^r_{2j} + \sum_{i=1}^{2} te_i = yp_1 + \max\{p_2, te_2 + t_2\} + 2\sum_{i=1}^{2} t_i + 2\sum_{i=1}^{2} te_i +$$
$$(j-y-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2\right) + p_2 = S^r_{1,j+1},$$

implies that there is no overlap between these two last operations. Lastly, after completing the move of J_n from M_1 to M_2 at

$$S^r_{1,y+1} + (n-y-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) + t_1 =$$
$$yp_1 + \sum_{i=1}^{2} t_i + \max\{p_2, te_2 + t_2\} + \sum_{i=1}^{2} te_i + (n-y-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) + t_1,$$

the robot performs its last move of J_n from M_2 to M_3 during time interval

$$(S^r_{2n}, C^r_{2n}] = (S^m_{2,y+1} + (n-y-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) + p_2, S^r_{2n} + t_2].$$

The fact that

$$yp_1 + \sum_{i=1}^{2} t_i + \max\{p_2, te_2 + t_2\} + \sum_{i=1}^{2} te_i + (n-y-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2\right) + t_1$$
$$\leq S^m_{2,y+1} + (n-y-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2\right) + p_2$$
$$= yp_1 + t_1 + \max\{p_2, te_2 + t_2\} + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + (n-y-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2\right) + p_2$$

implies that there is no overlap between these two last operations as well and Condition 3 holds.

The feasibility of Schedule 1.1.3.2(b) implies that the completion time of J_n on machine M_3 is at

$$C^m_{3n} = \max\{yp_1 + t_1 + t_2 + (n-y+1)p_3, C^m_{2n} + t_2\} + p_3 = yp_1 + \sum_{i=1}^{2} t_i + p_3 +$$

$$\max\{(n-y+1)p_3, t_1+\max\{p_2, te_2\}+t_2\}+\sum_{i=1}^{2} te_i+(n-y-1)(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i+p_2)+p_2+t_2\}.$$

The fact that this time matches the lower bound in eq. (18) when $(n-1)\left(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i+p_2-p_1\right) > \Delta(y^*)$ implies that this schedule 1.1.3.2(b) is optimal for subcase 1.1.3.2 when $y^* < n$. ■

4.5 Summary

This chapter presents a robotic three-machine flow-shop scheduling problem where jobs are identical and a single robot is responsible for the transportation of jobs between consecutive machines. The objective is to find a robot schedule which minimizes the makespan. As far as we know, only Hurink and Knust [38] consider a robotic flow-shop problem with more than two machines and with the objective of minimizing the makespan. They proved that the problem on $m \geq 2$ machines (m is arbitrary) is solvable in polynomial time if processing times are both machine- and job-independent and empty return times are negligible. Our analysis, although restricted to three-machines, relaxes their two assumptions by considering the more general case where (i) processing times are machine-dependent and job-independent, and (ii) empty return times are not necessarily negligible. We provide an efficient procedure to solve the problem by decomposing it into a set of sub-problems and provide a schedule for each sub-problem that matches the lower bound value.

5 A Combined Robot Selection and Scheduling Problem

In this section, we provide an analysis of the $\mathcal{RSSP}$, including its four variations $\mathcal{RSSP}_1 - \mathcal{RSSP}_4$ (see problem definition in Section 3.1.2 for an exact definition of the problem and its variations).[2] This section is divided into seven sections as follows. In the first section we provide necessary and sufficient conditions for a feasible schedule. In the second section we show that $\mathcal{RSSP}_2 - \mathcal{RSSP}_4$ are $\mathcal{NP}$-hard. Then, in the third section, we show that the $\mathcal{RSSP}$ reduces to a bicriteria shortest path problem within a directed acyclic graph. Based on this reduction, we were able to provide a polynomial time procedure to solve $\mathcal{RSSP}_1$ (in the fourth section), and exact and approximate algorithms for the solution of $\mathcal{RSSP}_2 - \mathcal{RSSP}_4$ (in the fifth section). In the sixth section, we provide some important special cases for which $\mathcal{RSSP}_2 - \mathcal{RSSP}_4$ are solvable in polynomial time. Lastly, in the seventh section, we summarize our results.

5.1 Necessary and sufficient conditions for a feasible schedule

Consider a single robot that serves the entire set of m machines. Due to the *Restrictions 1-2* (see Section 3.1.2), job J_j has to start its processing on machine M_m at time $(j-1)p + p(m-1) + \sum_{i=1}^{m-1} t_i$. Thus, for any feasible schedule, the makespan value which is equal to the completion time of job J_n on machine M_m is:

$$C_{\max} = p(m-1) + \sum_{i=1}^{m-1} t_i + np = \sum_{i=1}^{m-1} t_i + p(n+m-1). \tag{40}$$

Hurink and Knust [38] consider a similar problem with a single robot designated to serve the entire set of m machines. In contrast to $\mathcal{RSSP}$, they assume zero empty return times without a no-wait restriction. Hurink and Knust [38] provided a polynomial time algorithm that minimizes the makespan. Their algorithm can be described as follows: Let $Z = (z_1, ..., z_{n(m-1)})$ be a sequence of robot moves, where $z_\lambda = i$ implies that the λ'th transportation of the robot transfers a job from machine M_i to machine M_{i+1}. Note that since Hurink and Knust assume zero empty return times, a schedule can be defined only by robot transfer moves. For $i = 1, ..., m-1$, Hurink and Knust define special sequences of robot moves, $a^i = (1, ..., i)$ and $\widehat{a}^i = (i, ..., m-1)$, in which the robot serves the machines in the order $M_1, ..., M_i$ and $M_i, M_{i+1}, ..., M_{m-1}$, respectively. They show that the optimal robot

[2]The analysis in this section has been included in a research paper that was recently published in Omega [82].

schedule (i.e., the sequence of moves) includes only these two special sequences of robot moves. More particularly, the optimal robot schedule which minimizes the makespan is:

$$Z^* = (a^1, a^2, ..., a^{m-2}, a^{m-1}, a^{m-1}, ..., a^{m-1}, a^{m-1} = \widehat{a}^1, \widehat{a}^2, \widehat{a}^3, ..., \widehat{a}^{m-1}). \tag{41}$$

Let (J_j, M_i) represent an operation where a robot moves J_j from M_i to M_{i+1}. Also, substitute "J_j completes on M_i" for the operation "the robot returns empty to M_i and waits for the completion of job J_j (beside M_i) to start the next sequence of moves." The first part, $(a^1, a^2, ..., a^{m-2}, a^{m-1})$, which includes $m-1$ sequences of robot moves, is denoted as the *building-up phase* where in each step one more machine is served. At this phase, during sequence a^i the robot perform operations $(J_i, M_1), (J_{i-1}, M_2), ..., (J_1, M_i)$, then J_{i+1} completes on M_1. The second part, $(a^{m-1}, a^{m-1}, ..., a^{m-1})$, is denoted as the *identity phase* where all machines are served regularly. This phase includes $n-m$ sequences of moves each of which begins and ends with the robot beside machine M_1. During each sequence of moves, the robot moves a single job between any two consecutive machines until it places a job beside machine M_m and return empty to machine M_1 to begin the next move sequence. The last part, $(a^{m-1} = \widehat{a}^1, \widehat{a}^2, \widehat{a}^3, ..., \widehat{a}^{m-1})$, which includes $m-1$ sequences of robot moves, is denoted as the *building-down phase* where in each step one less machine is served. In this phase, during sequence $\widehat{a}^i$ $(i = 1, ..., m-1)$, the robot perform operations $(J_n, M_i), (J_{n-1}, M_{i+1}), ..., (J_{n-m+i+1}, M_{m-1})$, then (as long as $i \leq m-2$) J_n completes on M_{i+1} to begin the next sequence of moves $(\widehat{a}^{i+1})$.

Hurink and Knust show that if $\sum_{i=1}^{m-1} t_i \leq p$ then the algorithm yields a schedule in which machine M_s $(s > 1)$ starts to process the first job at time $(s-1)p + \sum_{i=1}^{s-1} t_i$ and processes the entire set of n jobs sequentially with no idle times. Thus, the n'th jobs completion time on machine M_s is $(s-1)p + \sum_{i=1}^{s-1} t_i + np$, and the makespan which equals to the completion time of the last job on machine M_m is equal to $(m-1)p + \sum_{i=1}^{m-1} t_i + np$ (equal to the makespan value in (40)).

5.1.1 Single robot scheduling with no-wait restrictions and non-zero empty return times

Let us consider the case of a single robot with *Restrictions 1-2* (see Section 3.1.2) and non-zero empty return times. Similar to Hurink and Knust we define a schedule by a set of $n(m-1)$ robot non-empty moves, $Z = (z_1, ..., z_{n(m-1)})$. Note that if $z_\lambda = i_2$ and $z_{\lambda+1} = i_1$ with $i_2 \geq i_1$, then between these two non-empty moves there is an empty return of the robot from machine M_{i_2+1} to machine M_{i_1} which requires $\sum_{i=i_1}^{i_2} te_i$ time.

Lemma 5 *For the case of a single robot, a schedule with a makespan equal to* $(m+n-1)p+\sum_{i=1}^{m-1} t_i$ *can be constructed iff* $\sum_{i=1}^{m-1}(t_i + te_i) \leq p$.

Proof. Let us first show that if $\sum_{i=1}^{m-1}(t_i + te_i) \leq p$, then a schedule with a makespan equal to $(m + n - 1)p + \sum_{i=1}^{m-1} t_i$ can be constructed. Given an instance with n jobs and m machines, if $\sum_{i=1}^{m-1}(t_i + te_i) \leq p$ a set of moves (see eq. (41)) defined by Hurink and Knust [38] can be made as follows: the robot applies each sequence of moves, with no delays until it reaches back machine M_1. Then the robot waits for the completion time of a job beside M_1 and starts the next move in a similar fashion (an illustration of this type of move appears in subsection (Appendix) 9.5.1). Since the time to complete sequence a^i within the *building-up phase* is $\sum_{r=1}^{i-1}(t_r + te_r)$, the robot waits to the end of this sequence for $p - \sum_{r=1}^{i-1}(t_r + te_r) \geq 0$ times units beside machine M_1 for the completion of job J_i on this machine. Since the time to complete sequence a^{m-1} within the *identity phase* is $\sum_{r=1}^{m-1}(t_r + te_r)$, the robot waits at the end of this sequence for $p - \sum_{r=1}^{m-1}(t_r + te_r) \geq 0$ times units beside machine M_1 before starting the next sequence of moves. Moreover, since the time to complete sequence $\widehat{a}^i$ within the *building-down phase* is $\sum_{r=i}^{m-1} t_r + \sum_{r=i+1}^{m-1} te_r$, the robot waits at the end of this sequence for $p - \sum_{r=i}^{m-1} t_r - \sum_{r=i+1}^{m-1} te_r > 0$ times units beside machine M_{i+1}. This implies that the robot return to each machine in cycles of p units of time. Thus, the production is done without delays and the completion time of J_n on M_m is at time $(m + n - 1)p + \sum_{i=1}^{m-1} t_i$.

It is now shown that if $\sum_{s=1}^{m-1}(t_s + te_s) > p$, then a schedule with a makespan equal to $(m+n-1)p+\sum_{i=1}^{m-1} t_i$ cannot be constructed. By contradiction, assume that $\sum_{s=1}^{m-1}(t_s+te_s) > p$ and a schedule with a makespan equals to $(m + n - 1)p + \sum_{i=1}^{m-1} t_i$ can be constructed. Consider the following two possible events. Event 1 where J_1 was transferred from machine M_{m-1} to machine M_m before J_m was transferred from machine M_1 to M_2. Event 2 is the opposite of the above.

Event 1: Here, if J_1 arrives at M_m later than time $p(m - 1) + \sum_{i=1}^{m-1} t_i$ then $C_{\max} > (m + n - 1)p + \sum_{i=1}^{m-1} t_i$. Therefore, J_1 arrives at M_m at time $p(m - 1) + \sum_{i=1}^{m-1} t_i$. Thus, the robot will be available to transfer J_m from M_1 to M_2 no earlier than time $p(m - 1) + \sum_{i=1}^{m-1} t_i + \sum_{i=1}^{m-1} te_i$, and J_m will be completed on M_m no earlier than time $p(m - 1) + \sum_{i=1}^{m-1} t_i + \sum_{i=1}^{m-1} te_i + \sum_{i=1}^{m-1} t_i + p(m - 1)$. As a result, J_n will be completed at M_m no earlier than time $p(m - 1) + \sum_{i=1}^{m-1} t_i + \sum_{i=1}^{m-1} te_i + \sum_{i=1}^{m-1} t_i + p(m - 1) + (n - m)p = p(m + m - 2) + \sum_{i=1}^{m-1} t_i + \sum_{i=1}^{m-1} te_i + \sum_{i=1}^{m-1} t_i > p(n + m - 1) + \sum_{i=1}^{m-1} t_i$. This contradicts our assumption that a schedule with a makespan value equal to $(m + n - 1)p + \sum_{i=1}^{m-1} t_i$ can be constructed.

Event 2: Here, J_m has been transferred from M_1 to M_2 before J_1 was transferred from M_{m-1} to M_m. Thus, J_1 is transfer from M_{m-1} to M_m no earlier than time $pm + \sum_{i=1}^{m-1} t_i$ and thus J_n will be completed on M_m no earlier $pm + \sum_{i=1}^{m-1} t_i + np$, which contradicts the assumption that a schedule with a makespan value equal to $(m+n-1)p + \sum_{i=1}^{m-1} t_i$ can be constructed. ■

The next corollary is straightforward from the fact that if we apply the set of moves in (41) then job J_j starts its processing on M_1 exactly at time $(j-1)p$ for $j = 1, ..., n$.

Corollary 1 *By applying the set of moves in (41) one can obtain a feasible schedule where J_j starts its processing on M_m at time $(m-1)p + \sum_{i=1}^{m-1} t_i + (j-1)p$ even if the arrival time of job J_j is a positive number not greater than $(j-1)p$.*

5.1.2 Multi-robot scheduling with no-wait restrictions and non-zero empty return times

Consider a multi-robot system comprised of k robots, $R_1, R_2, ..., R_k$. Let the machines be partitioned into k subsets $\mathcal{M}_r = \{M_{l_r}, ..., M_{l_{r+1}}\}$ $(r = 1, ..., k)$, and let R_r be responsible for transferring jobs between any pair of successive machines in set $\mathcal{M}_r$ be of type $\{r\}$. This implies that the total robot assignment cost is $TRC = \sum_{r=1}^{k} \delta_{\{r\}}$. Due to *Restrictions 1-2*, J_j has to start its processing on machine $M_s \in \mathcal{M}_r$ at time $(j-1)p + \sum_{f=1}^{r-1} \sum_{i=l_f}^{l_{f+1}-1} t_{i\{f\}} + \sum_{i=l_r}^{s-1} t_{i\{r\}} + p(s-1)$. Thus, the makespan value which is equal to the completion time of J_n on M_m is

$$C_{\max} = np + T(S) + p(m-1) = T(S) + p(m+n-1) \tag{42}$$

for any feasible schedule, where

$$T(S) = \sum_{r=1}^{k} \sum_{i=l_r}^{l_{r+1}-1} t_{i\{r\}}. \tag{43}$$

Note that the value of $p(m+n-1)$ is identical for all feasible solutions, while the value of $T(S)$ depends on the (feasible) assignment of robot to machines. Let $T(S)$ be the variable part of the makespan value in any feasible solution, S.

Below it is shown that the algorithm by Hurink and Knust [38] can be extended to provide a schedule with a makespan value shown in eq. (42) for our multi-robot system, if the following condition is satisfied for $r = 1, ..., k$:

$$\sum_{i=l_r}^{l_{r+1}-1} (t_{i\{r\}} + te_{i\{r\}}) \le p \tag{44}$$

Moreover, we show that if the condition in (44) is not satisfied then a schedule with a makespan value shown in (42) cannot be constructed.

Lemma 6 *For a multi-robot system, a feasible schedule with a makespan value shown in (42) can be constructed iff the condition in (44) holds.*

Proof. First it is shown that if the condition in (44) holds, then a feasible schedule with a makespan value shown in (42) can be constructed. Consider the sub-system that is served by robot R_1. According to Lemma 5, since $\sum_{i=l_1}^{l_2-1}(t_{i\{1\}} + te_{i\{1\}}) \le p$, if applying the set of moves in (41) on set $\mathcal{M}_1 = \{M_{l_1}, ..., M_{l_2}\}$ then J_j will start its processing on M_{l_2} at time $A_1 + (j-1)p$, where $A_1 = (l_2 - 1)p + \sum_{i=1}^{l_2-1} t_{i\{1\}}$ is the arrival time of J_j to the consecutive sub-system that is served by robot R_2. Thus, from Lemma 5 and Corollary 1, by applying the set of moves in (41) on the set of machines $\mathcal{M}_2 = \{M_{l_2}, ..., M_{l_3}\}$, job J_j will start its processing on machine M_{l_3} at time $A_2 + (j-1)p$ where $A_2 = (l_3 - 1)p + \sum_{f=1}^{2}\sum_{i=l_f}^{l_{f+1}-1} t_{i\{f\}}$, which is the arrival time of J_j to the consecutive sub-system that is served by robot R_3. Accordingly, it is easy to show that by applying the set of moves in (eq. 41) on the set of machines $\mathcal{M}_r = \{M_{l_r}, ..., M_{l_{r+1}}\}$ for $r = 3, ..., k$, J_j will start its processing on machine $M_{l_{r+1}}$ at time $A_r + (j-1)p$, where $A_r = (l_{r+1} - 1)p + \sum_{f=1}^{r}\sum_{i=l_f}^{l_{f+1}-1} t_{i\{f\}}$. This implies that for $r = k$ the processing of job J_n on $M_{k+1} = M_m$ starts at time $A_k + (n-1)p = (l_{k+1} - 1)p + \sum_{f=1}^{k}\sum_{i=l_f}^{l_{f+1}-1} t_{i\{f\}} + (n-1)p = (m-1)p + \sum_{f=1}^{k}\sum_{i=l_f}^{l_{f+1}-1} t_{i\{f\}} + (n-1)p$ and the makespan value which is the time that J_n is completed on $M_{k+1} = M_m$ is equal to $A_k + (n-1)p + p = (m-1)p + \sum_{f=1}^{k}\sum_{i=l_f}^{l_{f+1}-1} t_{i\{f\}} + np = (m+n-1)p + \sum_{f=1}^{k}\sum_{i=l_f}^{l_{f+1}-1} t_{i\{f\}}$. The fact that for a single robot problem if $\sum_{s=1}^{m-1}(t_s + te_s) > p$, then a schedule with a makespan equal to $(m-1)p + \sum_{i=1}^{m-1} t_i + np$ cannot be constructed (see Lemma 5) completes our proof. ■

5.2 $\mathcal{NP}$-hardness of problems $\mathcal{RSSP}_2 - \mathcal{RSSP}_4$

The $\mathcal{NP}$-completeness of the *DV* (See Definition 21 in Section 3.1.2) will be proven by showing that decision version of the $\mathcal{NP}$-complete *0-1 knapsack problem* (defined below) can be polynomial reduced to *DV*.

Definition 22 *Decision version of the 0-1 knapsack problem: Given two positive integers W and B and a finite set of n elements $\bar{A} = \{\bar{a}_1, \bar{a}_2, ..., \bar{a}_h\}$, where each element $\bar{a}_q$ has a weight of w_q and a benefit of b_q (w_q and b_q are positive integers) for $q = 1, ..., h$, determine whether there exists a subset $\bar{A}_1 \subseteq \bar{A}$ of elements such that $\sum_{\bar{a}_q \in \bar{A}_1} w_q \le W$ and $\sum_{\bar{a}_q \in \bar{A}_1} b_i \ge B$.*

Theorem 1 DV *is $\mathcal{NP}$-complete.*

Proof. Construct the following instance of the *DV* from an instance of the *0-1 knapsack problem*: There are $h+1$ machines, and $Q = 2h$ robot types. The job processing time is

$$p = \max_{q=1,...,h}\{b_q\} + 1. \tag{45}$$

Moreover, for $q = 1, ..., h$ we have that

$$\delta_{2q-1} = w_q + 1, \tag{46}$$

$$\delta_{2q} = 1, \tag{47}$$

$$t_{i,2q-1} = \begin{cases} 1 & \text{if } i = q, \\ p+1 & \text{otherwise.} \end{cases} \tag{48}$$

and

$$t_{i,2q} = \begin{cases} b_q + 1 & \text{if } i = q, \\ p+1 & \text{otherwise.} \end{cases} \tag{49}$$

The empty return times are all equal to zero, that is

$$te_{iq} = 0 \text{ for } i = 1, ..., h \text{ and } q = 1, ..., 2h, \tag{50}$$

and limitations are

$$\overline{C}_{\max} = h + \sum_{q=1}^{h} b_q - B + (n+m-1)p \text{ and } \overline{TRC} = W + h. \tag{51}$$

It is clear that the above transformation can be done in polynomial time. Note that according to Eqs. (48)-(50) the feasibility conditions that $\sum_{i=l_r}^{l_{r+1}-1}(t_{i,2q-1} + te_{i,2q-1}) \leq p$ and that $\sum_{i=l_r}^{l_{r+1}-1}(t_{i,2q} + te_{i,2q}) \leq p$ holds only for $l_r = q$ and $l_{r+1} = q+1$. Thus, robot types $2q-1$ and $2q$ can only be assigned to machine set $\{M_q, M_{q+1}\}$ for $q = 1, ..., h$, and only one of them has to be chosen to this set of machines.

It is first shown that if there is an instance for the *0-1 knapsack problem* which yields a *YES* answer, then there exists a solution for the *DV* problem with $C_{\max}(S) \leq h + \sum_{q=1}^{h} b_q - B + (n+m-1)p$ and $TRC(S) \leq W + h$. Given an instance which yields a *YES* answer for the *0-1 knapsack problem*, construct the following solution for the *DV* problem. For $q = 1, ..., n$, if $\bar{a}_q \in \bar{A}_1$ assign robot of type $2q-1$ to machine set $\{M_q, M_{q+1}\}$. Otherwise, assign robot of type $2q$ to this machine set. This solution is feasible since $t_{q,2q-1} + te_{q,2q-1} = 1 < \max_{q=1,...,n}\{b_q\} + 1 = p$ and $t_{q,2q} + te_{q,2q} = b_q + 1 \leq \max_{q=1,...,n}\{b_q\} + 1 = p$ for $q = 1, ..., n$.

Moreover,

$$TRC(S) = \sum_{\bar{a}_q \in \bar{A}_1} (w_q + 1) + \sum_{\bar{a}_q \in \bar{A} \setminus \bar{A}_1} 1 = \sum_{\bar{a}_q \in \bar{A}_1} w_q + \sum_{\bar{a}_q \in \bar{A}} 1 \leq W + h,$$

and

$$C_{\max}(S) = \sum_{\bar{a}_q \in \bar{A}_1} 1 + \sum_{\bar{a}_q \in \bar{A} \setminus \bar{A}_1} (b_q + 1) + (n + m - 1)p = h + \sum_{\bar{a}_q \in \bar{A} \setminus \bar{A}_1} (b_q) + (n + m - 1)p =$$
$$h + \sum_{q=1}^{h} b_q - \sum_{\bar{a}_q \in \bar{A}_1} b_q + (n + m - 1)p \leq h + \sum_{q=1}^{h} b_q - B + (n + m - 1)p.$$

Next it is shown that if there is a solution S for the *DV* problem with $C_{\max}(S) \leq h + \sum_{q=1}^{h} b_q - B + (n+m-1)p$ and $TRC(S) \leq W + h$, then a solution that yields a *YES* answer for the decision version of the *0-1 knapsack problem* problem can be found. Since solution S is feasible, machine set $\{M_q, M_{q+1}\}$ is assigned by either a robot of type $2q - 1$ or a robot of type $2q$ for $q = 1, ..., h$. Given solution S for the given instance of the *DV* problem, let S_1 be the set of all indices $q \in \{1, ..., n\}$ such that robot of type $2q - 1$ is assigned to machine set $\{M_q, M_{q+1}\}$. Let S_2 be the complementary set. Moreover, define the following solution for the decision version of the *0-1 knapsack problem* problem. For $q = 1, ..., h$, if $q \in S_1$, then set $\bar{a}_q \in \bar{A}_1$. The fact that $TRC(S) \leq W + h$ implies that

$$TRC(S) = \sum_{q \in S_1} (w_q + 1) + \sum_{q \in S_2} 1 = \sum_{q \in S_1} w_q + \sum_{q=1}^{h} 1 = \sum_{\bar{a}_q \in \bar{A}_1} w_q + h \leq W + h,$$

and thus

$$\sum_{\bar{a}_q \in \bar{A}_1} w_q \leq W. \tag{52}$$

Moreover, the fact that $C_{\max}(S) \leq h + \sum_{q=1}^{h} b_q - B + (n + m - 1)p$ implies that

$$C_{\max}(S) = \sum_{q \in S_1} 1 + \sum_{q \in S_2} (b_q + 1) + (n + m - 1)p = \sum_{q=1}^{h} 1 + \sum_{q \in S_2} b_q + (n + m - 1)p =$$

$$h + \sum_{q=1}^{h} b_q - \sum_{q \in S_1} b_q + (n+m-1)p = h + \sum_{q=1}^{h} b_q - \sum_{a_q \in A_1} b_q + (n+m-1)p \leq h + \sum_{q=1}^{h} b_q - B + (n+m-1)p,$$

results in

$$\sum_{\bar{a}_q \in \bar{A}_1} b_q \geq B. \tag{53}$$

The fact that Eq. (52) and eq. (53) hold implies a *YES* answer for the decision version of the *0-1 knapsack problem.* ■

The following corollary is now straightforward from Theorem 1 and the fact that $\mathcal{RSSP}_4$ is at least as hard as $\mathcal{RSSP}_2 - \mathcal{RSSP}_3$.

Corollary 2 *Problems $\mathcal{RSSP}_2 - \mathcal{RSSP}_4$ are NP-hard.*

5.3 A polynomial reduction of $\mathcal{RSSP}$ to a bicriteria $\mathcal{SPP}$

In this section it is shown that the bicriteria $\mathcal{RSSP}$ can be reduced to a bicriteria $\mathcal{SPP}$ within a directed acyclic multigraph. This reduction will enable the construction of various algorithms to solve the different variations of $\mathcal{RSSP}$. First a formal definition of a directed acyclic multigraph is given, and then the $\mathcal{SPP}$ within a directed acyclic multigraph and its four different variations are presented.

Definition 23 *A directed acyclic multigraph $G(V, E)$, where $V = \{1, ..., N\}$ is the set of nodes and E is the set of arcs, is a graph in which between any two nodes u to v with $v > u$ there is a set of arcs $E(u, v) = \{(u, v)_1, ..., (u, v)_{n_{(u,v)}}\} \in E$ (which might be an empty set) each of which is directed from u to v.*

In the bicriteria $\mathcal{SPP}$, each arc $(u, v)_g \in E(u, v)$ in the multigraph is associated with two non-negative integer parameters. The first $d_{(u,v)_g}$ is the *duration* and the second $c_{(u,v)_g}$ is the *cost* of arc $(u, v)_g$ $(g = 1, ..., n_{(u,v)})$. Let $D(P) = \sum_{q=1}^{k} d_{(l_q,l_{q+1})_{[q]}}$ be the total duration and $C(P) = \sum_{q=1}^{k} c_{(l_q,l_{q+1})_{[q]}}$ be the total cost of a path $P = \{(l_1 = 1, l_2)_{[1]}, (l_2, l_3)_{[2]}, ..., (l_k, l_{k+1} = N)_{[k]}\}$ from vertex 1 to vertex N in $G(V, E)$ where l_j is the jth node in path P and $[j]$ is the index of the arc in set $E(l_j, l_{j+1})$ that belongs to path P. Similar to $\mathcal{RSSP}$, four different variations of the bicriteria $\mathcal{SPP}$ on a directed acyclic multigraph are defined:

- $\mathcal{SPP}_1$: find a path P in $G = (V, E)$ which minimizes $D(P) + C(P)$.
- $\mathcal{SPP}_2$: find a path P in $G = (V, E)$ which minimizes $D(P)$ subject to: $C(P) \leq \overline{C}$ where $\overline{C}$ is a given upper bound on the total cost.
- $\mathcal{SPP}_3$: find a path P in $G = (V, E)$ which minimizes $C(P)$ subject to: $D(P) \leq \overline{D}$ where $\overline{D}$ is a given upper bound on the total duration.

- $\mathcal{SPP}_4$: Identify a Pareto-optimal solution for each Pareto-optimal point.

Theorem 2 *$\mathcal{RSSP}$ is polynomial reducible to $\mathcal{SPP}$.*

Proof. Construct the following instance to $\mathcal{SPP}$ from an instance of $\mathcal{RSSP}$: set $N = m$ (a single node in the multigraph is included for each machine in the $\mathcal{RSSP}$) and for any combination of a robot type q ($q = 1, ..., Q$) and machines M_u and M_v ($1 \leq u \leq m-1$ and $u+1 \leq v \leq m$). If $\sum_{i=u}^{v-1}(t_{iq}+te_{iq}) \leq p$, then include an arc $(u,v)_q$ in set E which is directed from u to v and corresponds to the feasible assignment of a robot of type q to serve machines $M_u, ..., M_v$. Two values are associated with each arc $(u,v)_q \in E$. The first

$$d_{(u,v)_q} = \sum_{i=u}^{v-1} t_{iq}, \tag{54}$$

is the *duration* of arc $(u,v)_q$ and the second

$$c_{(u,v)_q} = \delta_q, \tag{55}$$

is the *cost* of arc $(u,v)_q$ (the above polynomial reduction is illustrated in subsection (Appendix) 9.5.2). Note that each path $P = \{(l_1 = 1, l_2)_{\{1\}}, (l_2, l_3)_{\{2\}}, ..., (l_k, l_{k+1} = N)_{\{k\}}\}$ in $G(V,E)$ corresponds to a feasible assignment of robot to machines, where a robot of type $\{r\}$ is assigned to serve machine set $\{M_{l_r}, ..., M_{l_{r+1}}\}(r = 1, ..., k)$. Moreover, the duration and cost of each arc, $d_{(l_r,l_{r+1})_{\{r\}}}$ and $c_{(l_r,l_{r+1})_{\{r\}}}$, corresponds to the addition to the makespan value and the total robot cost resulting from the (feasible) assignment of a robot of type $\{r\}$ to serve machine set $\{M_{l_r}, ..., M_{l_{r+1}}\}$ ($r = 1, ..., k$). Below, it is shown that there is a path P in $G(V,E)$, with $D(P) \leq \overline{D}$ and $C(P) \leq \overline{C}$ *iff* there is a solution for $\mathcal{RSSP}$ with $C_{\max}(S) \leq \overline{D} + (n+m-1)p$ and $TRC(S) \leq \overline{C}$.

First assume that there exists a feasible solution S for $\mathcal{RSSP}$ with $C_{\max}(S) \leq \overline{D} + (n+m-1)p$ and $TRC(S) \leq \overline{C}$, where $\mathcal{M}_r = \{M_{l_r}, ..., M_{l_{r+1}}\}$ is the subset of machines that are served by robot of type $\{r\}$ ($r = 1, ..., k$). The fact that S is a feasible solution implies that $\sum_{i=l_r}^{l_{r+1}-1}(t_{i\{r\}} + te_{i(r\}}) \leq p$ for $r = 1, ..., k$. Thus, according to the above transformation, path $P = \{(l_1 = 1, l_2)_{\{1\}}, (l_2, l_3)_{\{2\}}, ..., (l_k, l_{k+1} = N)_{\{k\}}\}$ exists in $G(V,E)$. Since $C_{\max}(S) \leq \overline{D} + (n+m-1)p$, then $C_{\max}(S) = \sum_{r=1}^{k}\sum_{i=l_r}^{l_{r+1}-1} t_{i,\{r\}} + (n+m-1)p \leq \overline{D} + (n+m-1)p$, and therefore also that $\sum_{r=1}^{k}\sum_{i=l_r}^{l_{r+1}-1} t_{i,\{r\}} = \sum_{r=1}^{k} d_{(l_r,l_{r+1})_{\{r\}}} \leq \overline{D}$. Moreover, since $TRC(S) \leq \overline{C}$, we have $TRC(S) = \sum_{r=1}^{k} \delta_{\{r\}} = \sum_{r=1}^{k} c_{(l_r,l_{r+1})_{\{r\}}} \leq \overline{C}$, which implies that for path $P = \{(l_1 = 1, l_2)_{\{1\}}, (l_2, l_3)_{\{2\}}, ..., (l_k, l_{k+1} = m)_{\{k\}}\}$ it is true that $\sum_{r=1}^{k} d_{(l_r,l_{r+1})_{\{r\}}} \leq \overline{D}$ and $\sum_{r=1}^{k} c_{(l_r,l_{r+1})_{\{r\}}} \leq \overline{C}$.

Second, it is shown that if there is no solution for $\mathcal{RSSP}$, with $C_{\max}(S) \leq \overline{D}+(n+m-1)p$ and $TRC(S) \leq \overline{C}$; then there is no path P in $G(V,E)$ with $D(P) \leq \overline{D}$ and $C(P) \leq \overline{C}$. By contradiction, assume that there is a path $P = \{(l_1 = 1, l_2)_{\{1\}}, (l_2, l_3)_{\{2\}}, ..., (l_k, l_{k+1} = N)_{\{k\}}\}$ in $G(V,E)$, with $D(P) \leq \overline{D}$ and $C(P) \leq \overline{C}$. Given path P, construct a solution S for $\mathcal{RSSP}$ by assigning a type $\{r\}$ robot to serve machines in set $\mathcal{M}_r = \{M_{l_r}, ..., M_{l_{r+1}}\}$ $(r = 1, ..., k)$. Solution S is feasible since, according to our polynomial transformation, we have that $\sum_{i=l_r}^{l_{r+1}-1}(t_{i\{r\}} + te_{i(r\}}) \leq p$ $(r = 1, ..., k)$. Moreover, according to (54) $D(P) \leq \overline{D}$ implies $D(P) = \sum_{r=1}^{k} d_{(l_r,l_{r+1})_{\{r\}}} = \sum_{r=1}^{k}\sum_{i=l_r}^{l_{r+1}-1} t_{i,\{r\}} \leq \overline{D}$. Also, according to (55) $C(P) \leq \overline{C}$ implies that $C(P) = \sum_{r=1}^{k} c_{(l_r,l_{r+1})_{\{r\}}} = \sum_{r=1}^{k} \delta_{\{r\}} \leq \overline{C}$. Therefore, $C_{\max}(S) = \sum_{r=1}^{k}\sum_{i=l_r}^{l_{r+1}-1} t_{i,\{r\}} + (n+m-1)p \leq \overline{D} + (n+m-1)p$, and $TRC(S) = \sum_{r=1}^{k} \delta_{\{r\}} \leq \overline{C}$, which results in a contradiction. ■

5.4 Polynomial time procedure for $\mathcal{RSSP}_1$

For solving $\mathcal{RSPP}_1$, construct the following simple directed acyclic graph, $G(\overline{V}, \overline{E})$, from the multi-graph $G(V,E)$. Set $\overline{V} = V$. Calculate the *weight*, $w_{(u,v)_q}$, of each arc $(u,v)_q \in E(u,v)$ by

$$w_{(u,v)_q} = d_{(u,v)_q} + c_{(u,v)_q}. \tag{56}$$

If $E(u,v) \neq \emptyset$, include a single arc (u,v) in $\overline{E}$ which is directed from u to v. The length of this arc is defined by the minimum arc weight among all arcs that are directed from u to v. That is,

$$l_{(u,v)} = \min_{q \in E(u,v)} \left\{ w_{(u,v)_q} \right\}. \tag{57}$$

The resulting graph $G(\overline{V}, \overline{E})$ is a directed acyclic graph and the length of the shortest path in $G(\overline{V}, \overline{E})$ corresponds to the minimum objective value of $\mathcal{RSPP}_1$. The shortest path in $G(\overline{V}, \overline{E})$ can be determined in $O(m^2)$ time by applying the following simple recursion:

$$G_v = \min_{(u,v) \in \overline{E}} \left(G_u + l_{(u,v)} \right), \tag{58}$$

where $G_1 = 0$ is the initial condition, and G_v is the length of the shortest path that ends in v in the subgraph that includes the vertices $0, ..., v$. Thus, G_m corresponds to the length of the shortest path in $G(\overline{V}, \overline{E})$.

Based on the above analysis and the polynomial transformation that appears in the proof of Theorem 2, the following optimization algorithm to solve $\mathcal{RSSP}_1$ is presented (the implementation of the algorithm on a numerical example is illustrated by in subsection

(Appendix) 9.5.3).

Algorithm 3 *The optimization algorithm for solving $\mathcal{RSSP}_1$.*

```
Input: $m$, $p$, $\boldsymbol{\delta} = (\delta_1, ..., \delta_Q)$, $\mathbf{t} = (t_{iq})$ and $\mathbf{te} = (te_{iq})$ for $i = 1, ..., m$ and $q = 1, ..., Q$.
Step 1. For any combination of a robot type $q$ ($q = 1, ..., Q$) and machines $M_u$ and $M_v$
        ($1 \leq u < v \leq m$), if $\sum_{i=u}^{v-1}(t_{iq} + te_{iq}) \leq p$, then include robot type $q$ in set
        $E(u, v)$.
Step 2. Construct the digraph $G(\overline{V}, \overline{E})$ as follows: Set $\overline{V} = \{1, ..., m\}$. For any
        $1 \leq u < v \leq m$, if $E(u, v) \neq \emptyset$ then include a single directed arc $(u, v)$ in set $\overline{E}$.
Step 3. Calculate the length of each arc $(u, v) \in \overline{E}$ by (57) where $w_{(u,v)_q}$ is
        calculated by Eqs. (54)-(56). Set $q_{(u,v)} = \arg\min_{q \in A(u,v)} \{w_{(u,v)_q}\}$ where $q_{(u,v)}$
        is the robot type which corresponds to the shortest arc's length.
Step 4. Find the shortest path in $G(\overline{V}, \overline{E})$ by applying the recursion (58) for
        $v = 1, ..., m$ with the initial condition $G_1 = 0$.
Step 5. Trace back the shortest path in graph $G(\overline{V}, \overline{E})$ to identify its internal vertices
        $l_1 = 1, l_2, l_3, ..., l_k, l_{k+1} = m$. Assign robot of type $q_{(l_r, l_{r+1})}$ to serve machine set
        $\mathcal{M}_r = \{M_{l_r}, ..., M_{l_{r+1}}\}$ ($r = 1, ..., k$) and schedule the robots according to the
        set of moves in (41).
```

Theorem 3 *Algorithm 3 solves $\mathcal{RSSP}_1$ in $O(m^2Q)$ time.*

Proof. Since there are $O(m^2Q)$ combinations of a robot type q ($q = 1, ..., Q$) and machines M_u and M_v ($1 \leq u < v \leq m$), Step 1 requires $O(m^2Q)$ time and Step 2 requires $O(m^2)$ time. Calculating the length of each arc by (57) in Step 3 requires $O(Q)$ time. Since there are $O(m^2)$ arcs in $G(\overline{V}, \overline{E})$, Step 3 can be done in $O(m^2Q)$ time. Step 4 requires the implementation of the recursion in (58) for $v = 1, ..., m$ which requires $O(m^2)$ time. Since Step 5 can be implemented in $O(m)$ time, the overall computational complexity of the algorithm is $O(m^2Q)$. ■

5.5 Exact and approximate algorithms for $\mathcal{RSSP}_2 - \mathcal{RSSP}_4$

Hassin [37] provides exact algorithms to solve $\mathcal{SPP}_2$ and $\mathcal{SPP}_3$, on a simple directed acyclic graph, in a pseudo-polynomial time algorithm of $O(|E|\,\overline{C})$ and $O(|E|\,\overline{D})$, respectively. Moreover, he presented a fully polynomial time approximation scheme ($FPTAS$) that runs in $O(|E|\,(N^2/\varepsilon)\log(N/\varepsilon))$ time. The fact that $\mathcal{SPP}_2$ and $\mathcal{SPP}_3$ on a directed acyclic *multi-graph* can be reduced to a similar problem on a simple directed acyclic graph which includes $O\left(\sum_{u=2}^{N}\sum_{v=1}^{u-1} n_{(u,v)}\right)$ nodes, implies that the algorithms provided by Hassin [37] can be used to provide a pseudo-polynomial time algorithms and a *FPTAS* for $\mathcal{SPP}_2$ and $\mathcal{SPP}_3$ on a multigraph as well.

Below, the pseudo-polynomial time algorithm provided by Hassin [37] is extended to directly solve $\mathcal{SPP}_3$ on a multigraph (instead of reducing it to an equivalent problem on a simple graph). Let $G(V, E)$ be a directed acyclic multigraph with $V = \{1, ..., N\}$ and $E = \{E(u, v)\,|u < v\}$ where $E(u, v) = \{(u, v)_1, ..., (u, v)_{n_{(u,v)}}\}$ is a set of $n_{(u,v)}$ directed arcs from node u to node v with $1 \leq u < v \leq N$. Let $d_{(u,v)_q}$ and $c_{(u,v)_q}$ be positive integers representing the duration and the cost of any arc $(u, v)_q$ for $1 \leq u < v \leq N$ and $q = 1, ..., n_{(u,v)}$.

Let $f(v, D)$ be the minimum cost path from vertex 1 to vertex v, whose duration is not greater than D. The value of $f(v, D)$ for $v = 2, ..., N$ and $D = 1, ..., \overline{D}$ can be computed by applying the following recursion:

$$f(v, D) = \min \begin{cases} f(v, D-1) \\ \min\limits_{1 \leq u < v,\ d_{(u,v)_q} \leq D,\ q=1,...,n_{(u,v)}} \left\{ f\left(u, D - d_{(u,v)_q}\right) + c_{(u,v)_q} \right\} \end{cases} \tag{59}$$

with the initial conditions

$$f(1, D) = 0 \text{ for } D = 0, ..., \overline{D} \text{ and } f(v, 0) = \infty \text{ for } v = 2, ..., m. \tag{60}$$

The optimal solution value is then given by $f(N, \overline{D})$, and the optimal path is determined by tracking the path back from the end. It is easy to verify that by applying the above recursion the $\mathcal{SPP}_3$ problem can be solved in $O\left(\overline{D} \sum_{u=2}^{N} \sum_{v=1}^{u-1} n_{(u,v)}\right) = O(|E|\,\overline{D})$ time. Moreover, a similar recursion can be applied to solve the $\mathcal{SPP}_2$ problem in $O(|E|\,\overline{C})$ time.

Let D_{UB} be an upper bound on the duration of any path in the graph. The value of D_{UB} can be determined in $O(m^2)$ time by applying the following recursion for $v = 2, ..., N$:

$$f_v = \max_{0 \leq u < v} \left\{ f_u + \overline{d}_{(u,v)} \right\} \tag{61}$$

with the initial condition that $f_1 = 0$, where $\overline{d}_{(u,v)} \overset{def}{=} \max\limits_{q=1,...,n_{(u,v)}} \left\{ d_{(u,v)_q} \right\}$. After applying the recursion, the actual value of D_{UB} is given by f_N. Then, by setting $\overline{D} = D_{UB}$, the recursion is used for solving $\mathcal{SPP}_3$, can be used to solve $\mathcal{SPP}_4$ in a pseudo-polynomial time of $O(|E|\,D_{UB})$.

Based on the above analysis and the polynomial transformation that appears in the proof of Theorem 2, the following optimization algorithm to solve $\mathcal{RSSP}_3$ is provided.

Algorithm 4 *An optimization algorithm for solving $\mathcal{RSSP}_3$.*

Input*: m, p, $\boldsymbol{\delta} = (\delta_1, ..., \delta_Q)$, $\mathbf{t} = (t_{iq})$ and $\mathbf{te} = (te_{iq})$ for $i = 1, ..., m$ and $q = 1, ..., Q$.*

Step 1. *Construct the multigraph $G(V, E)$ as follows: Set $V = \{1, ..., m\}$. For any combination of a robot type q ($q = 1, ..., Q$) and M_u and M_v ($1 \leq u < v \leq m$) if $\sum_{i=u}^{v-1}(t_{iq} + te_{iq}) \leq p$, then include an arc $(u, v)_q$ in set E in set E (which is directed from u to v). Calculate the duration and weight of arc $(u, v)_q$ by (54) and (55).*

Step 2. *Apply the recursion in (eq. 59) for $v = 2, ..., N$ and $D = 1, ..., \overline{D}$ with the initial conditions in (eq. 60). The optimal solution value is given by $f(N, \overline{D})$.*

Step 3. *Trace back the optimal path in graph $G(V, E)$ to identify its internal vertices $(l_1 = 1, l_2)_{\{1\}}, (l_2, l_3)_{\{2\}}, ..., (l_k, l_{k+1} = m)_{\{k\}}$. Assign robot of type $\{r\}$ to serve machine set $\mathcal{M}_r = \{M_{l_r}, ..., M_{l_{r+1}}\}$ ($r = 1, ..., k$) and schedule the robots according to the set of moves in (eq. 41).*

Theorem 4 *Algorithm 4 solves $\mathcal{RSSP}_3$ in $O(m^2 Q\overline{D})$ time.*

Proof. Constructing the multigraph $G(V, E)$ in Step 1 requires $O(m^2 Q)$ time. Since there are $O(m^2 Q)$ arcs in the multigraph, Step 2 can be applied in a $O(|E|\,\overline{D}) = O(m^2 Q\overline{D})$ time. The fact that Step 3 can be implemented in $O(m)$ time completes our proof. ■

It is easy to observe that Algorithm 4 can be modified to solve problems $\mathcal{RSSP}_2$ and $\mathcal{RSSP}_4$ in pseudo-polynomial time as well. To conclude this section, it is emphasized that any algorithm that solves any variant of $\mathcal{SPP}$ on a simple (non-multi) graph can be converted to solve the same variant on a multigraph. Thus, one can easily use any method to solve any variant of $\mathcal{SPP}$ to solve the corresponding variant of $\mathcal{RSSP}$. Among the possible algorithms, one can find a set of fully polynomial time approximation schemes (FPTAS's) as the ones presented by Hassin [37], Lorenz and Raz [66] and Ergun *et al.* [24]. For further information about various methods to solve $\mathcal{SPP}$, see a recent survey paper by Garroppo *et al.* [28].

5.6 Special cases of $\mathcal{RSSP}_2 - \mathcal{RSSP}_4$

In this section we present three special cases where $\mathcal{RSSP}_2 - \mathcal{RSSP}_4$ are solvable in polynomial time.

5.6.1 The case of robot independent transportation times

Consider the case where transportation times are robot-independent, that is, $t_{iq} = t_i$ for $i = 1, ..., m - 1$. This implies that robots differ only by their cost and empty return times. For this case, according to eq. (54), the duration of each arc $(u, v)_q$ for the equivalent $\mathcal{SPP}$ problem is given by $d_{(u,v)_q} = \sum_{i=u}^{v-1} t_{iq} = \sum_{i=u}^{v-1} t_i$. Thus, for any path $P = \{(l_1 =$

$1, l_2)_{\{1\}}, (l_2, l_3)_{\{2\}}, ..., (l_k, l_{k+1} = N)_{\{k\}}\}$ from vertex 1 to vertex N in $G(V, E)$, the total duration is given by $D(P) = \sum_{r=1}^{k} d_{(l_r,l_{r+1})_{\{r\}}} = \sum_{r=1}^{k} \sum_{i=l_r}^{l_{r+1}-1} t_{i\{r\}} = \sum_{r=1}^{k} \sum_{i=l_r}^{l_{r+1}-1} t_i = \sum_{i=1}^{N-1} t_i$ which is path independent. Thus, there is a single Pareto-optimal point for the equivalent instance of the $\mathcal{SPP}$ which corresponds to the minimum cost path. Since the equivalent instance of the $\mathcal{SPP}$ may have more than a single directed arc connecting any two nodes (u, v) with $1 \leq u < v \leq N$, each of which has the same duration of $\sum_{i=u}^{v-1} t_i$, among all those arcs keep a single arc (u, v) which has the minimum cost among all arcs in set $E(u, v)$. The cost of any arc (u, v) with $1 \leq u < v \leq N$ is given by

$$c_{(u,v)} = \min_{q \in E(u,v)} \left\{ c_{(u,v)_q} \right\}. \tag{62}$$

The resulting graph $G(\overline{V}, \overline{E})$ is a simple (non-multi) directed acyclic graph. The shortest cost path in $G(\overline{V}, \overline{E})$ can be determined in $O(m^2)$ time by applying the following simple recursion:

$$G_v = \min_{(u,v) \in \overline{E}} \left(G_u + c_{(u,v)} \right), \tag{63}$$

where $G_1 = 0$ is the initial condition, and G_v is the length of the shortest cost path that ends in v in the subgraph that includes the vertices $0, ..., v$. Thus, G_m corresponds to the value of the shortest cost path in the graph.

Based on the above analysis and the polynomial transformation that appears in the proof of Theorem 2, the following optimization algorithm for solving $\mathcal{RSSP}_4$, for the special case where the transportation times are robot-independent can be constructed.

Algorithm 5 *The optimization algorithm for solving $\mathcal{RSSP}_4$ with robot-independent transportation times.*

```
Input: $m$, $p$, $\boldsymbol{\delta} = (\delta_1, ..., \delta_Q)$, $\mathbf{t} = (t_i)$ for $i = 1, ..., m$ and $\mathbf{te} = (te_{iq})$ for $i = 1, ..., m$
        and $q = 1, ..., Q$.
Step 1-2. Apply Steps 1-2 of Algorithm 3.
Step 3. Calculate the cost of each arc $(u, v) \in \overline{E}$ by (62) where $c_{(u,v)_q}$ is
        calculated by (55). Set $q_{(u,v)} = \arg\min_{q \in E(u,v)} \{c_{(u,v)_q}\}$ which is the robot type that
        corresponds to the minimum arc cost.
Step 4. Find the minimum cost path in $G(\overline{V}, \overline{E})$ by applying the recursion in (63) for
        $v = 1, ..., m$ with the initial condition $G_1 = 0$.
Step 5. Apply Step 5 of Algorithm 3.
```

Theorem 5 *Algorithm 5 solves the $\mathcal{RSSP}_4$ problem with robot-independent transportation times in $O(m^2 Q)$ time.*

Proof. According to the proof of Theorem 3, Steps 1, 2 and 5 require $O(m^2Q)$, $O(m^2)$ and $O(m)$ time, respectively. Moreover, since the cost calculating of each arc by (62) in Step 3 requires $O(Q)$ time, and there are $O(m^2)$ arcs, Step 3 can be done in $O(m^2Q)$ time. The proof is completed by fact that Step 4 requires the implementation of the recursion in (63) for $v = 1, ..., m$ which requires $O(m^2)$ time. ■

5.6.2 The case of identical robot costs

Consider a case where all robots have the same cost, that is, $\delta_q = \delta$ for $q = 1, ..., Q$. For this case, according to (55), the cost of each arc in the equivalent instance of the $\mathcal{SPP}$ equals to δ, i.e., it is robot-independent. Since the equivalent instance of the $\mathcal{SPP}$ may have more than a single directed arc connecting any two nodes (u, v) with $1 \leq u < v \leq N$, each of which has the same cost, among all those arcs keep a single arc (u, v) (directed from u to v) that has the minimum duration among all arcs in set $E(u, v)$. That is, the duration of each arc (u, v) with $1 \leq u < v \leq N$ is given by

$$d_{(u,v)} = \min_{q \in E(u,v)} \left\{ d_{(u,v)_q} \right\}. \tag{64}$$

The resulting $\mathcal{SPP}$ is now defined on a simple graph $G(\overline{V}, \overline{E})$ rather than on the multi-graph $G(V, E)$, as each set of arcs $E(u, v)$ is replaced by a single arc. Note, however, that unlike the previous special case, although each arc has the same cost, the total cost of each path is still path-dependent. More particularly, the cost of each path $S = \{(l_1 = 1, l_2)_{\{1\}}, (l_2, l_3)_{\{2\}}, ..., (l_k, l_{k+1} = N)_{\{k\}}\}$ from vertex 1 to vertex N in $G(\overline{V}, \overline{E})$ is equal to $k\delta$ where k is the number of arcs (selected robots) in the path. Since $k \in \{1, 2, ...N - 1\}$, there are at most $N - 1$ possible Pareto-optimal points for the equivalent instance of the $\mathcal{SSP}_4$.

Let $G_v(k)$ be the shortest duration path that ends in v in the subgraph that includes vertices $0, ..., v$ with exactly k arcs. The value of $G_v(k)$ can be determined by applying the following recursion for $1 \leq k < v \leq N$:

$$G_v(k) = \min_{(u,v) \in \overline{E}, u \geq k} \left(G_u(k-1) + d_{(u,v)} \right), \tag{65}$$

where $G_1(0) = 0$ is the initial condition. After computing all $G_v(k)$ values, for every $\overline{k} \in \{1, 2, ...N - 1\}$, a Pareto-optimal point is given by the pair $(G_N(k), k)$, with the minimum $G_N(k)$ value among all $G_N(k)$ values with $k = 1, ..., \overline{k}$.

Based on the above analysis and the polynomial transformation that appears in the proof of Theorem 2, the following optimization algorithm is constructed to solve $\mathcal{RSSP}_4$ for the special case of equal cost robots.

```
Algorithm 6 The optimization algorithm for solving $\mathcal{RSSP}_4$ with identical robot costs.
    Input: $m$, $p$,$\delta$, $\mathbf{t} = (t_{iq})$ and $\mathbf{te} = (te_{iq})$ for $i = 1, ..., m$ and $q = 1, ..., Q$.
    Step 1-2. Apply Steps 1-2 of Algorithm 3.
    Step 3. Calculate the duration of each arc $(u, v) \in \overline{E}$ by (64), where $d_{(u,v)_q}$ is
            calculated by (54). Set $q_{(u,v)} = \arg \min_{q \in E(u,v)} \{d_{(u,v)_q}\}$ which is the robot
            type that corresponds to the minimum arc's duration.
    Step 4. Apply the recursion in (65) for $1 \leq k < v \leq N$ with the initial condition
            $G_1(0) = 0$.
    Step 5. For every $\overline{k} \in \{1, 2, ...N-1\}$ a Pareto-optimal point is given by the pair $(G_N(k), k)$
            with the minimum $G_N(k)$ value among all $G_N(k)$ values with $k = 1, ..., \overline{k}$.
    Step 6. For each Pareto-optimal point, $(G_N(k), k)$, trace back the shortest duration path
            in graph $G(\overline{V}, \overline{E})$ to identify its internal vertices $l_1 = 1, l_2, l_3, ..., l_k, l_{k+1} = m$.
            Assign robot of type $q_{(l_r, l_{r+1})}$ to serve machine set $\mathcal{M}_r = \{M_{l_r}, ..., M_{l_{r+1}}\}$
            $(r = 1, ..., k)$ and schedule the robots according to the set of moves in (41).
```

Theorem 6 *Algorithm 6 solves the $\mathcal{RSSP}_4$ with identical robot costs in $O(m^2 \max\{m, Q\})$ time.*

Proof. According to the proof of Theorem 3, Steps 1 and 2 require $O(m^2Q)$ and $O(m^2)$ time, respectively. Calculating the duration of each arc by (64) in Step 3 requires $O(Q)$ time. Since there are $O(m^2)$ arcs, Step 3 can be done in $O(m^2Q)$ time. Step 4 requires the implementation of the recursion in (65) $(1 \leq k < v \leq N = m)$ which requires $O(m^3)$ time. Step 5 can be implemented in $O(m)$ time, which is also the time complexity of implementing Step 6 for each Pareto-point. Since there are $O(m)$ Pareto-optimal solutions, the time complexity of Step 6 is $O(m^2)$ and, thus, the overall computational complexity of the algorithm is $O(\max\{m^2Q, m^3) = O(m^2 \max\{Q, m\})$. ■

5.6.3 The case of a single robot type

For simplicity, in this subsection we omit the robot index q, such that, for example, t_i is the transportation time from M_i to M_{i+1} made by the single robot and δ is the robot cost. Since there is a single robot type $(Q = 1)$, the multigraph reduces to a simple graph $G(V, E)$. Moreover, according to eqs. (42) and (54), the total duration of any feasible path $S = \{(l_1 = 1, l_2)_{\{1\}}, (l_2, l_3)_{\{2\}}, ..., (l_k, l_{k+1} = N)_{\{k\}}\}$ in the equivalent instance of the $\mathcal{SPP}$ problem is given by $D(S) = \sum_{i=1}^{N-1} t_i$. This directly implies that the duration of each path is path-independent, which further implies that the makespan value of *any* feasible solution for the $\mathcal{RSSP}$ is equal to $D(S) + (n + m - 1)p = \sum_{i=1}^{N-1} t_i + (n + m - 1)p$ (the makespan value is independent of the decision of how to assign robots to machines). Moreover, according to (55) the cost of each arc in the graph is identical to the robot cost, δ. Thus, our objective is to find a path in $G(V,E)$ with the minimum number of arcs. Following the result in Lemma 6, a single robot can *cover* machines $\{M_u,...,M_v\}$ if $\sum_{i=u}^{v-1}(t_{iq}+te_{iq}) \leq p$. Moreover, the *maximal cover*, $\breve{a}_u$, of a single robot that is assigned to serve a set of machines that begins in machine M_u can be calculated by finding the maximal v value for which the condition that $\sum_{i=u}^{v-1}(t_i+te_i) \leq p$ holds. That is

$$\breve{a}_u = \arg\max_{v=u+1,...,m+1}\left\{\sum_{i=u}^{v-1}(t_i+te_i)\,\middle|\,\sum_{i=u}^{v-1}(t_i+te_i)\leq p\right\}. \tag{66}$$

The following lemma will be used subsequently to construct an $O(m\log m)$ time algorithm to solve the $\mathcal{RSSP}_4$ with a single robot type.

Lemma 7 *For each Pareto-optimal point, there exists a Pareto-optimal solution such that if a robot is assigned to serve machine set* $\{M_{l_r},...,M_{l_{r+1}}\}$ *then* $\breve{a}_{l_r}=l_{r+1}$.

Proof. Consider a Pareto-optimal solution S, where k robots are assigned to serve the m machines and robot R_r is assigned to serve machine set $\{M_{l_r},...,M_{l_{r+1}}\}$ $(r=1,...,k)$. Moreover, assume that $\bar{r}$ is the maximal robot index r such that $\breve{a}_{l_{\bar{r}}} \neq l_{\bar{r}+1}$. Since solution S is feasible $\breve{a}_{l_{\bar{r}}} > l_{\bar{r}+1}$ and thus $\sum_{i=l_{\bar{r}}}^{l_{\bar{r}+1}-1}(t_i+te_i) \leq \sum_{i=l_{\bar{r}}}^{\breve{a}_{l_{\bar{r}}}-1}(t_i+te_i) \leq p$. Moreover, the fact that $m \geq \breve{a}_{l_{\bar{r}}}$ implies that there is a consecutive set of machines $\{M_{l_{\bar{r}+1}},...,M_{l_{\bar{r}+2}}\}$ which is served by robot $R_{\bar{r}+1}$. Since S is a feasible solution we have that $\sum_{i=l_{\bar{r}+1}}^{l_{\bar{r}+2}-1}(t_i+te_i) \leq p$. Define an alternative solution $\widetilde{S}$ where robot $R_{\bar{r}}$ is assigned to machine set $\{M_{l_{\bar{r}}},...,M_{\breve{a}_{l_{\bar{r}}}}\}$ and robot $R_{\bar{r}+1}$ is assigned to machine set $\{M_{\breve{a}_{l_{\bar{r}}}},...,M_{l_{\bar{r}+2}}\}$ while all other robots are assigned to the same machine set as in S. Solution $\widetilde{S}$ is feasible since $\sum_{i=l_{\bar{r}}}^{\breve{a}_{l_{\bar{r}}}-1}(t_i+te_i) \leq p$ and $\sum_{i=\breve{a}_{l_{\bar{r}}}}^{l_{\bar{r}+2}-1}(t_i+te_i) \leq \sum_{i=l_{\bar{r}+1}}^{l_{\bar{r}+2}-1}(t_i+te_i) \leq p$. Moreover, since the makespan value is independent of the solution and the robot cost of both solutions S and $\widetilde{S}$ is equal to $k\delta$, solution $\widetilde{S}$ is Pareto-optimal as well. The lemma now follows from the fact that the proof can be applied repeatedly to obtain a Pareto-optimal solution, where $\breve{a}_{l_r}=l_{r+1}$ for each robot R_r that is assigned to serve machine set $\{M_{l_r},...,M_{l_{r+1}}\}$. ■

Since any path begins in the first node representing the first machine (i.e., the first robot has to be assigned to a subset of machines that begins in machine M_1), then according to Lemma 7, if $\breve{a}_1=\breve{a}_{l_1}=l_2$, then there exist a Pareto-optimal solution for which the first robot is assigned to machine set $\{M_{l_1},...,M_{l_2}\}$. Then, the second robot has to be assigned to a machine set that begins with machine M_{l_2}. According to Lemma 7, if $\breve{a}_{l_2}=l_3$, then there exist a Pareto-optimal solution for which the second robot is assigned to machine set

$\{M_{l_2}, ..., M_{l_3}\}$. In a similar way, based on Lemma 7, if $\breve{a}_{l_r} = l_{r+1} \leq m$, then there exist a Pareto-optimal solution for which the r'th robot is assigned to machine set $\{M_{l_r}, ..., M_{l_{r+1}}\}$. Thus, based on the above analysis and the polynomial transformation that appears in the proof of Theorem 2, the following optimization algorithm is constructed to solve $\mathcal{RSSP}_4$ for the special case of a single robot type (the algorithm is illustrated by a numerical example in subsection (Appendix) 9.5.4).

Algorithm 7 *The optimization algorithm for solving $\mathcal{RSSP}_4$ with a single robot type.*

```
Input: $m$, $p$, $\boldsymbol{\delta}$, $\mathbf{t} = (t_i)$ and $\mathbf{te} = (te_i)$ for $i = 1, ..., m$.
Initialization: Set $l_1 = 1$ and $j = 1$.
Step 1. Calculate $\breve{a}_{l_j}$ by Eq. (66). Set $l_{j+1} = \breve{a}_{l_j}$ and assign a robot
        to set $\{M_{l_j}, ..., M_{l_{j+1}}\}$.
Step 2. If $l_{j+1} = m$ then stop. Otherwise, set $j = j + 1$ and go back to Step 1.
```

Theorem 7 *Algorithm 7 solves the $\mathcal{RSSP}_4$ with a single robot type in $O(m \log m)$ time.*

Proof. The calculation of any $\breve{a}_{l_j}$ value in Step 1 requires $O(\log m)$ time. Since $\breve{a}_{l_j} \geq l_j + 1$, Step 1 is repeated $O(m)$ times, and thus the algorithm indeed takes $O(m \log m)$ time. ■

5.7 Summary

A flow-shop scheduling problem with a no-wait restriction, for which robots are responsible for the transfer of jobs between any pair of consecutive machines was described. The robot move on a single track positioned alongside a machine transfer line. The objective was to (a) select a set of robots, (b) assign each robot to a portion of the track, and (c) define a set of moves for each robot, in order to minimize the makespan and robot selection costs. Four different variations of the problem were studied. The first is to find a solution that minimizes the sum of makespan and total robot selection cost. The second was to minimize the makespan subject to an upper bound on the value of the total robot selection cost. The third was to minimize the total robot selection cost subject to an upper bound on the makespan value. The last was to identify a Pareto optimal solution for each Pareto-optimal point with regard to the dual criteria. The last three variations were proven to be $\mathcal{NP}$-hard. Moreover, it was shown that the problem can be reformulated as a bicriteria shortest path problem within a directed multigraph. Based on this reformulation, a polynomial time procedure to solve the first variant was provided. It was shown that the three other variants can be solved in pseudo-polynomial time. In addition, three important special cases are derived for each of the three $\mathcal{NP}$-hard variants, were shown to be solved in polynomial time.

6 Single Robot Scheduling Using *RA RL* Collaboration

This section presents a collaborative *RL* method (Q-learning) for solving *Problem 1* (see subsection 3.1.1), where the robot can operate either autonomously (no adviser) or semi-autonomously (with adviser).[3]

6.1 *RL* formulation – states, actions, and rewards

This section presents the formulation of the robotic scheduling problem as an *RL* problem (states, actions, and rewards). System transition epochs are defined whenever the robot has just deposited a job at a machine and is free to carry out the next job transfer. At this time, an action is selected to define which machine the robot should next serve.

6.1.1 System state space

At transition epoch t, the state of the system is denoted as s_t, where s_t is defined by m possible buffer-machine states and the robot's location. At any time t there are four possible states for machine M_i and its output buffer OB_i $(i = 1, 2, \ldots, m)$: (1) OB_i is empty and machine M_i is idle, (2) OB_i is empty and machine M_i is busy, (3) OB_i is not empty and machine M_i is idle, (4) OB_i is not empty and machine M_i is busy. The IB_i is not part of the system state definition, since the machine and output buffer status provide enough information on whether the robot should consider serving a machine or not. Thus, the system state space S is defined by the states of the buffer-machine pairs and the location of the robot (Lr). Each system state is an $m+1$ vector $S = [s_1, s_2, \ldots, s_j, \ldots, s_m, Lr]$ with $s_j \in \{1, 2, 3, 4\}$ and $Lr \in \{1, 2, \ldots, m\}$. The initial state is $S_0 = [2, 1, \ldots, 1, 1]$ as all output buffers are empty and all machines are idle except for the first machine, which has started to process the first job (loaded automatically from IB_1). The final state is $S_{\check{T}} = [1, 1 \ldots 3, m]$ which occurs when all jobs have completed processing and are placed in the final output buffer OB_m. At each transition epoch t, a state is defined as $s_t \in S$.

6.1.2 Action space

The action space A is a set of $m-1$ actions $A = [a_1, a_2 \ldots a_i \ldots a_{m-1}]$ where action a_i is defined as a robot job transfer from M_i's OB_i to input buffer IB_{i+1} of the next machine M_{i+1} in the processing sequence. An action a_i is determined to be allowable (or not allowable) according to the state of machine M_i, i.e., if the state of the machine is $s_t \in \{2, 3, 4\}$ then

[3]The analysis in this chapter has been presented at the 21st International Conference on Production Research [85].

an action to serve machine M_i is allowable. Otherwise, $s_t \in \{1\}$ and an action to serve machine M_i is not allowable. In general, each job must be transferred by the robot through $m-1$ machines (unloading from machine M_m is automatic) and so the minimum number of transitions (and so actions) from the initial state to the final state is $n(m-1)$. More than $n(m-1)$ transitions (actions) are possible if the state-action policy is non-optimal.

System state transition times occur at the time the robot deposits a job at one of the machine input buffers. A job transition involves two successive robot movements. Assume the robot is at M_k and job J_j is to be moved from M_i to M_{i+1}. The first move, if $k > i$, takes $\sum_{g=i}^{g=k-1} te_g$ time (if $k < i$ takes $\sum_{g=k}^{g=i-1} te_g$ time, and if $k = i$ takes no time) from the robot's last drop off position at IB_k to pick up J_j. The second move takes time t_i from OB_i to IB_{i+1}. However, it is possible that the second move will be delayed if there are no finished jobs residing in M_i's output buffer. In this case, the robot must wait until J_j will finish processing in M_i, i.e., until C_{ij}^m. Thus, in this case, the robot will wait for time $C_{ij}^m - (t + \sum_{g=i}^{g=k-1} te_i)$. Assume the system state has just changed and the time is t. At this time the robot has deposited a job at M_k, and the action is taken to transfer a job J_j from M_i to M_{i+1}. If M_i is idle, the transfer will be completed at time $t + \sum_{g=i}^{g=k-1} te_i + t_i$. If M_i is busy and J_j is being processed on M_i, the transfer will be completed t_i time units after J_j will finish processing in M_i, i.e., at time $C_{ij}^m + t_i$. For each case, the transfer will be at time τ:

$$\tau = \max\{t + \sum_{g=i}^{g=k-1} te_i, C_{ij}^m\}. \tag{67}$$

This implies that the system transition time will be $t + \sum_{g=i}^{g=k-1} te_i + t_i$ if the robot arrives after J_j is finished, and $C_{ij}^m + t_i$ if the robot arrives before J_j is finished processing on M_i. So, assuming the system starts at time 0, the system transition times can be defined as $t^1, t^2, ..., t^v, t^{v+1}$ with the system dynamics defined by $t^{v+1} = \tau$ for all $v = 0, 1, 2, ...$ At each of these times the system enters a new state S.

6.1.3 Rewards

The rewards, denoted by r_{ti}, are calculated at each transition time t and M_i $i = 1, 2, ..., m-1$ where $s_i \neq 1$, by

$$r_{ti} = t + \sum_{g=i}^{g=k-1} te_i - \min\{JC_i\}, \tag{68}$$

where $\min\{JC_i\}$ is the minimum completion time of J_j on machine M_i that is next to be transferred to M_{i+1}, i.e., completion time of J_j on machine M_i that is the "first" to be transferred to M_{i+1}. The "first" job to be transferred from M_i to $M_{i+1}$is: (i) if OB_i is empty then $\min\{JC_i\}$ is the completion time of J_j which is being processed on machine M_i, and (ii) if OB_i is not empty then $\min\{JC_i\}$ is the completion time of J_j that has not been transferred M_{i+1} and been waiting for the robot maximum time.

Q value and action selection method - A ε-greedy method of action selection is used, where ε is decreased over time according to (69) to encourage exploration at the beginning and exploitation as the robot improves by

$$\varepsilon = 1/(\acute{E}_i)^{\bar{\beta}}. \tag{69}$$

Here ε is the probability of selecting random action ($0 \leq \varepsilon \leq 1$), $\bar{\beta}$ is a positive number ($\bar{\beta} \geq 0$) that indicates how fast ε will decrease towards 0 and $\acute{E}_i$ is the episode i from the current learning session. At each time step t, a number is sampled from a uniform distribution between $[0, 1]$ and then compared to the value of ε. If this number is greater than ε, the robot will behave greedily (see subsection 1.3.1). However, if the number is smaller than ε the robot will perform an action with an equal probability. For any $\bar{\beta} \geq 0$, as $|\acute{E}_i|$ increases ε will decrease and the chance to perform a random action will be smaller, until $\varepsilon \longrightarrow 0$ and then all remaining actions are chosen greedily. The value of the system at state s_t when selecting action a_t, is $Q(s_t, a_t)$ and is updated as follows:

$$Q(s_t, a_t) = Q(s_t, a_t) + \bar{\alpha}[r_{t+1} + \bar{\gamma} \max Q(s_{t+1}, a_t) - Q(s_t, a_t)], \tag{70}$$

where $\bar{\alpha}$ is the learning rate that controls how much weight is given to the reward just experienced, and $\bar{\gamma}$ is the discount rate for future Q-values. The Q-value equation is updated for each transition $t = t^1, t^2, ..., t^v, t^{v+1}$. The transitions are terminated for the current episode when the system reaches its final state $S_{\check{T}} = [1, 1, \ldots, 3, m]$, which occurs when all jobs have completed processing and the robot has finished its work. If this event (i.e., $S_{\check{T}}$) does not occur after $\acute{E}$ learning episodes the process is terminated. The performance of the robot is the resulting makespan value, denoted as $C_{\max}(\acute{E}_i)$.

6.2 *RL* formulation – *RA* collaboration

The collaborative learning process between the robot and the adviser uses a modified version of Q-learning. The adviser is an agent that operates external to the system, and is simulated with different expertise levels. Two different situations were analyzed: (a) an adviser with fixed expertise, and (*b*) an adviser with increasing fatigue (deterioration of expertise as a function of time). The robot can operate either autonomously (AO) or semi- autonomously (SAO). In the SAO mode the robot collaborates with an adviser. Switching between modes is under the control of the robot that senses the degree of improvement or deterioration in performance. When the improvement level falls below a threshold the robot requests advice from the adviser. In addition, the robot is endowed with the ability to evaluate the assistance provided by the adviser and retains the option to accept it or not. The process, in general (Fig. 7), begins with independent learning by the robot. During the learning process the robot tests its performance and acts accordingly. When performance is high, the system stays in AO. When performance is low, the system switches to SAO and collaborates with the adviser. When operating SAO the collaborating adviser provides advice. If the advice is good it is accepted, otherwise it is rejected. If the robot considers the adviser "unfit" (i.e., as one that provides bad advice) the system switches permanently to the AO mode.

6.2.1 Robot - ε-greedy action selection

This subsection presents an algorithm that uses the basic concepts of RL based on ε-greedy action selection. In this method the robot behaves greedily (see subsection 1.3.1) most of the time, and from time to time (with probability ε) selects a random action [87]. At the beginning of the learning process, when the robot does not have much information, ε is relatively high in order to encourage exploration by the robot by allowing it to perform random actions. As the learning process progresses and the robot obtains more information, the value of ε decreases. The robot, in this state, will reduce the number of random actions and begin to exploit the information already gathered. Eq. (69) shows how ε decreases over time.

The robot is equipped with an ability to evaluate its learning performance. Learning performance is measured by the average learning performance measure, C_{ave}, calculated over the X most recent learning episodes, by (71):

$$C_{ave} = \sum_{i=j-X+1}^{i=j} C_{\max}(\acute{E}_i)/\ X, \tag{71}$$

where X is a predefined constant, and $C_{\max}(\acute{E}_i)$ is the completion time of J_n on M_n (i.e., C_{mn}^{m}) that was achieved at the end of $\acute{E}_i$, i.e., learning episode i. The measure C_{ave} is calculated at the end of the j'th episode (i.e., $\acute{E}_j$), and enables the robot to decide when to operate AO or SAO by comparing C_{ave} to an adaptive threshold $\hat{C}_{\max}$, which represents the minimum average completion time achieved so far from the beginning of the learning trial. The decision, if and when the robot should collaborate with an adviser, is determined according to the following rules. At first, set the initial threshold $\hat{C}_{\max} = C_{\max}(\acute{E}_1)$ (at the end of the first learning episode). Then, after X iterations compute the average learning performance measure C_{ave} by using Eq. (71). If $C_{ave} > \hat{C}_{\max}$ the robot changes to SAO mode and collaborates with the adviser. Otherwise, if $C_{ave} \leq \hat{C}_{\max}$ the robot continues in AO mode and sets $\hat{C}_{\max} = C_{ave}$.

The robot is also equipped with an ability to evaluate the adviser's advice. There are two types of adviser assessment. In a short-term advice assessment, the current advice is assessed by the robot to accept it or reject it. In long-term assessment, a cumulative record of the short-term acceptance rate is used to estimate the skill level of the adviser. When it is deemed that the adviser is "unfit" to give advice, then no additional advice is requested and the robot remains in autonomous mode and discontinues future collaboration. The robot, in the process of collaboration, should recognize the adviser's expertise and act accordingly. Levels of adviser expertise are represented by the parameter $Temp$ (Boltzmann distribution with values of $Temp = 0.01 - 1$) where $Temp = 0.01$ is considered an expert and $Temp = 1$ is considered a novice ([30]).

Short-term advice assessment - At first, the robot should decide whether to accept or reject adviser advice. This decision depends on the advice quality compared to the performance of the robot's learning process. When the advice (entire robot schedule) results in a better solution than the one achieved from the learning process, i.e., $\hat{C}_{\max} > C_{\max}(\text{adviser})$, the advice is considered as good advice and so will be received. Otherwise, the advice is considered as bad advice if $\hat{C}_{\max} \leq C_{\max}(\text{adviser})$, and so will be rejected.

Long-term advice assessment - Determined by using a scaled threshold (H_a). The scale, H_a, between $[0, 1]$ is divided into novice $[0, 1/3)$, medium $[1/3, 2/3)$, or expert $[2/3, 1]$. Exponential smoothing as defined in Eq. (72) was used to weight present advice relative to total accumulated advice.

$$H_{ai} = \alpha' D_i + (1 - \alpha') H_{a,i-1}, H_{a0} = 1, \tag{72}$$

where H_{a0} is the initial condition, $\alpha' (0 \leq \alpha' \leq 1)$ is the smoothing constant, which represents the importance of the present i'th advice, and $(1 - \alpha')$ is the importance of all past advice. $D_i = 0$ or 1 is the success of adviser's i'th advice instance, and H_{ai} $(0 \leq H_{ai} \leq 1)$ is the weighted total advice over all advice instances up to the i'th instance - adviser performance threshold.

An important question is how the robot decides when to discard a bad adviser and switch permanently to autonomous mode. Let θ be a random number sampled from a normal distribution $N\tilde{}(0.5, (1/6)^2)$, which is compared to an advice performance threshold (H_{ai}). When the ratio of adviser success (H_{ai}) is low, the probability to continue collaborating in SAO mode is low, and vice versa. For $\mu = 0.5$ and $\sigma = 1/6$, for a novice, the probability to sample a number (θ) that is larger than the value of a novice ratio is high ($0.84 - 1$), and thus the robot will be in AO mode most of the time. For a medium expert, the probability the sampled number θ is larger than the value of a medium ratio is medium ($0.16 - 0.84$) and so the robot at times will switch to SAO mode. For an expert, the probability that sampled number θ is larger than the value of the expert's ratio is very low ($0 - 0.16$) and so the robot will mostly switch to SAO mode.

The collaboration process, in general, is illustrated in Fig 7.

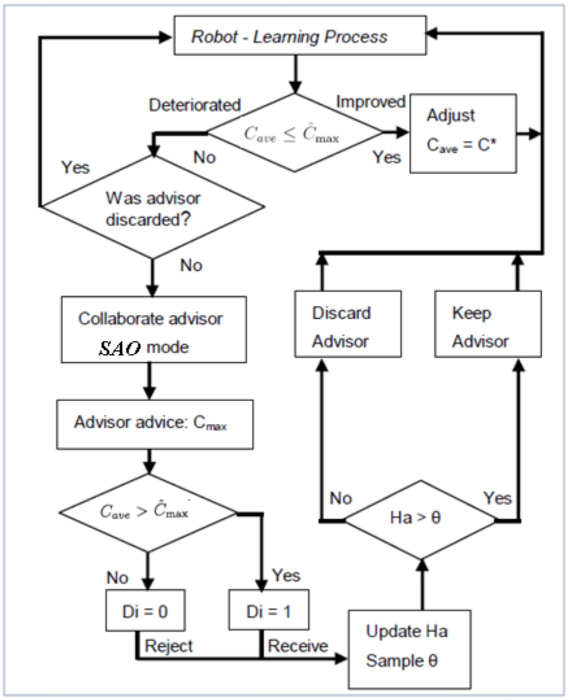

Figure 7: Flow diagram of RA collaboration.

6.2.2 Adviser – softmax action selection

The adviser's action selection behavior is modeled through a Boltzmann distribution in a softmax action selection operation, in which an action a is selected with probability $Prob_t(a_i)$ as shown in eq. (2). The skill level of the adviser is represented by the value of $Temp$, based on an optimal (or near optimal) Q-table (achieved by performing an enormous number of learning episodes) for the adviser simulation. Q_t is calculated according to ($s_t \in S$) and ($a_t \in A$), and $Temp$ is a factor that determines the amount of exploration. High $Temp$ values will result in a lot of exploration (equal probability for all actions) and low values will result in exploitation corresponding to the experience level. At the limit as $Temp \to 0$, softmax action selection becomes the same as the greedy action selection, but as $Temp$ grows softmax selection gives all actions equal probability. The adviser skill level is adjusted through the parameter $Temp$. An adviser with perfect skills is represented by low $Temp$ values while an adviser with poor skills is represented by a high $Temp$ value.

This research analyzes two different situations. The first is an adviser with fixed expertise, i.e., fixed value of $Temp$, and the second is an adviser with increasing fatigue. To evaluate adviser's skill level deterioration over time due to fatigue, the $Temp$ value is dynamically changed. At the beginning of the episode, the adviser with low $Temp$ is considered to provide good advice (expert), while during simulation the value of $Temp$ is increased so that the quality of advice deteriorates until reaching that of a novice. $Temp$ is dynamically changed as follows:

$$Temp = t_{sys} \ln(NAR), \tag{73}$$

where NAR is the number of times advice was requested from the adviser (if $NAR = 1$ we assign $NAR = 1.001$), and t_{sys} is the time (additive) of the system (i.e., learning session). Jointly, t_{sys} and NAR define how fast $Temp$ will increase.

6.3 *RA* collaboration experiments

The *RA RL* collaboration algorithms were programmed in *Matlab* (pseudo-code can be found in subsection (Appendix) 9.6). We evaluate the quality of the *RA RL* collaboration algorithm by two criteria: the makespan value and the ability of the robot to evaluate adviser expertise. Performance of the robotic schedule (i.e., makespan $C_{\max}$) was conducted by computing an upper bound (UB) error. Similar to Stern and Vinter ([84]), a UB error was calculated, as described in equation (74):

$$\textit{UB error} = (\min\{UB\} - \{LB\}) / \max\{LB\}, \tag{74}$$

where $\max\{LB\} = LB$ (see eq. 5) for a given problem, and $\min\{UB\}$ is the best result achieved by the *RA RL* collaboration algorithm. In order to evaluate the robot's cognitive abilities we have defined different levels of adviser expertise. This allows us to get an indication of how successful the robot was in recognizing adviser expertise, which affects the robot's behavior through a collaboration process.

Similar to [94], which first used RL for $F2 \,||\, C_{\max}$ (a problem that is solvable in $O(n \log n)$ time [39]), and then for $F3 \,||\, C_{\max}$ (a problem that is strongly $\mathcal{NP}$-hard ([27]), *RA RL* collaborative algorithm was tested first for a problem where machines are identical (*IM*, i.e., $p_{ij} = p$), and then for the more difficult problem, where machines are non-identical (*NIM*, i.e., $p_{ij} = p_i$).

The following subsection describes the simulation setup (subsection 6.3.1) and the results (subsection 6.3.2) for a single robot scheduling problem formulated as a collaborative *RA RL* problem.

6.3.1 Simulation setup

Tests were conducted for $m = 4$ machines and $n = 6$ jobs. In each simulation 10 trials were conducted, each consisting of $\acute{E} = 50$ learning episodes (this was empirically set based on a-priori analysis which indicated convergence after approximately 40 learning episodes). Job processing times are sampled from a uniform distribution between $[150, 200]$, and adjacent machine transfer times t_i (and empty return times te_i) are sampled uniformly between $[10, 20]$. For the *IM* case the process time must be sampled only once, while for the *NIM* case m samples are required, one for each machine M_i. The adjacent machine transfer time is sampled once for each trial and used additively to determine empty transfer times between non-adjacent machines. The following values were used for the reinforcement learning trials: $\bar{\beta} = 0.5$ (to encourage exploration), $\bar{\alpha} = 0.05$ (the learning rate), $\bar{\gamma} = 0.9$ (discount rate) as define by Kartun *et al.* [41], and $\alpha' = 0.2$ in order to give more weight to the value of accumulated advice and less weight to current advice. The cases of *IM* and *NIM* were tested with different levels of expertise represented by fixed values of $Temp$ varying over the range $(0.01 - 1)$. Additionally, a third case was tested for an adviser with a dynamic decreasing level of expertise (an adviser with fatigue) where $Temp$ changes according to eq. (73).

6.3.2 Results and discussion

Simulations were conducted with the level of adviser expertise as a parameter. The fixed levels of $Temp$ were $0.01, 0.25.0.50, 0.75$ and 1.0, where 1.0 represents a novice and 0.01 represents an expert. Tables 3 and 4 present the solutions for the flow-shop robotic problem for *IM* and *NIM*, respectively. The "Robot" and "Adviser" columns present solutions as *[X1-X2],X3*. The *[X1-X2]* is the **range of errors** where *X1* is the smallest error and *X2* is the biggest error. *X3* presents the **average error**. Errors where calculated by $\max\{LB\} = LB$ (eq. (5)) and according to eq. (74).

Table 3 indicates that the robot found an optimal solution for each value of $Temp$ within an average error of 1.8%. The biggest error (*UB error*) of the robot was 5.5% (simulation 4). The adviser, on the other hand, found an optimal solution only for expert values corresponding to $Temp = 0.01$ and $Temp = 0.25$. The biggest error (*UB error*) of the adviser was 23.3% when $Temp = 1$ (the adviser is novice). The errors of the adviser were much bigger than the robot with maximum average error of 15.5%.

Simulations	$Temp$	**Robot**	**Adviser**	**Number of times advice was requested**
1	0.01	[0 - 0], 0	[0 - 0], 0	[4 -9], 6.21
2	0.25	[0 - 0.041], 0.007	[0 – 0.108], 0.07	[3 - 7], 4.8
3	0.5	[0 - 0.033], 0.008	[0.038-0.165], 0.103	[3 - 7], 4.3
4	0.75	[0 - 0.055], 0.018	[0.062- 0.199], 0.126	[2 -5], 3.6
5	1	[0 - 0.039], 0.012	[0.092- 0.233], 0.155	[1 - 4], 2.6

Table 3: IM: Intervals and average errors for fixed T

Table 4 shows that the robot was capable of achieving an optimal solution for each expertise value of $Temp$ when the machines were non-identical. The maximum average error is 9.4%. The biggest error (*UB error*) of the robot was 23.1% (simulation 2), indicating that the robot once in a while can be misled when implementing advice that was considered to be "good" at the beginning of learning (i.e., in the exploration phase). The adviser, on the other hand, found an optimal solution only for $Temp = 0.01$. The greatest error (*UB error*) of the adviser was 27.9% when $Temp = 1$ (the adviser is novice). The errors of the adviser were much larger than that of the robot where the maximum average error of adviser was 12.7% (simulation 5) and robot with the adviser was 9.4% (simulation 4).

Simulations	$Temp$	**Robot**	**Adviser**	**Number of times advice was requested**
1	0.01	[0 - 0.094], 0.028	[0 - 0.094], 0.028	[5 -9], 6.8
2	0.25	[0 - 0.231],0.062	[0.011-0.263], 0.08	[4 -7], 5.1
3	0.5	[0 - 0.118],0.041	[0.024 -0.264], 0.096	[3 - 7], 4.3
4	0.75	[0 - 0.202],0.094	[0.08 -0.269], 0.112	[3 -6], 4
5	1	[0 - 0.161], 0.05	[0.088- 0.279], 0.127	[1 - 4], 2.8

Table 4: NIM: Intervals and average errors for fixed T

Tables 3 and 4 indicate that the average error and its range for the adviser increases as the value of $Temp$ increases for both the *IM* (Table 3) and *NIM* (Table 4) cases. The robot found the optimal solution for at least one trial for each simulation. The adviser, on the other hand, obtained worse results as the value of $Temp$ increased. Furthermore, it can be seen that the value of $Temp$ fits the number of times advice was requested, requesting more collaborations with an expert and much less with a novice.

It can be seen from the table that the robot found better solutions than the adviser (except for $Temp = 0.01$). In general, solutions achieved by the robot and adviser, for the *IM* case, were better than for the *NIM* case. It can be seen that as the value of T increases the average *UB error* and intervals also increase. An interesting point is that when $Temp = 0.75$ (for both *IM* and *NIM*) the robot had the maximum average *UB error*, and when $Temp = 1$ the average *UB error* is slightly smaller. Tables 3 and 4 also indicate that the average number of times advice was requested increases as $Temp$ decreases, which implies that the robot collaborates more with an expert adviser than with a novice adviser. The results show that the robot was able to operate correctly and switch control between AO and SAO. It is important to note that upon convergence of the *RL* algorithm of the best solution obtained through the collaboration process, the collaborative robot-adviser system's maximum average error was 9.4% (see Table 3, simulation 1).

More simulations were conducted for dynamic levels of expertise due to "fatigue". Tables 5 and 6 indicate that in the case of a fatiguing adviser the robot was able to distinguish between the adviser's expertise and acted accordingly. When the adviser is an expert ($Temp = 0.01$) there is more collaboration, and when the adviser's skill level deteriorates (i.e., a novice, $Temp = 1$) collaboration is minimal.

Table 5 presents the intervals of errors and average error.

Simulations	**Machines**	**Robot**	**Adviser**
1	*IM*	[0 - 0], 0	[0 - 0], 0
2	*NIM*	[0 - 0.236], 0.074	[0 - 0.236],0.08

Table 5: Dynamic level of expertise - Results

Table 6 presents important variables that refer to the collaboration process.

Simulations	**Machines**	**H_a**	**Number of times advice was requested**
1	*IM*	[0.327 - 0.64]	[3 -6], 4.7
2	*NIM*	[0.328 - 0.64]	[3 - 7], 4.1

Table 6: Dynamic level of expertise - Collaboration

Both robot and adviser perform better for the *IM* case than the *NIM* case. The intervals of Ha indicate the values of Ha at the time that the adviser last collaborated, i.e., Ha is the time point when the adviser was discarded. For *IM* and *NIM*, the minimum value of Ha was 0.327 and 0.328 and the maximum was 0.64 and 0.64, respectively. Thus, the robot collaborated for the last time with a medium or a novice adviser according to the scale of Ha.

6.4 Summary

In conclusion, the robot overcomes two significant problems: misjudging an expert and learning from a novice. Optimal robot schedules were obtained for *IM* and *NIM* problems, and for both fixed and dynamic levels of expertise. The adviser, on the other hand, provided optimal solutions only for small values of $Temp$ (*IM*: $Temp = 0.01$, 0.25 and *NIM*: $Temp = 0.01$), which represented expertise. Furthermore, as the value of $Temp$ increases (representing lower levels of expertise) the average number of times advice was requested decreases. This indicates that the cognitive abilities of the robot in detecting the credibility of adviser advice functioned properly.

The algorithm based on ε-greedy action selection was shown to be capable of providing an optimal solution for at least one trial on each simulation for both the *IM* and *NIM*, and for both fixed and dynamic levels of expertise. Implementation of collaboration in the scheduling problem provides, in some cases, optimal solutions and also accelerates the learning process of the robot. On the other hand, it can also mislead the robot and divert it from achieving the optimal solution.

The formulation and solution methodology presented here is not limited to robots confined to a fixed linear track, but is also appropriate for problems where the machines are located anywhere in free space and job transfers are carried out by a mobile robot.

7 Two Robots Scheduling Using *RR RL* Collaboration

This section presents a collaborative *RL* method (*Q*-learning) for a two-robot scheduling problem (*Problem 3*, see subsection 3.1.3), where robots of the same or different type can operate in one of four levels of collaboration: *Full*, *Pull*, *Push*, and *None*.[4]

7.1 *RL* formulation – states, actions, and rewards

This section presents the formulation of *Problem 3* as a collaborative *RL* problem (states, actions, and rewards). System transition epochs are defined whenever one of the robots has just deposited a job at a machine and is free to carry out the next job transfer. At this time, an action is selected to define which machine this robot should next serve.

7.1.1 System state space

At transition epoch t, the state of the system is denoted by the states of each of the m machines. Let $s_i \in \{1, 2, 3, 4\}$ be the state of machine M_i ($i = 1, ..., m$), where the states are: (1) OB_i is empty and machine M_i is idle, (2) OB_i is empty and machine M_i is busy, (3) OB_i is occupied and machine M_i is idle, and (4) OB_i is occupied and machine M_i is busy. The state of the system at time epoch t is denoted as $S_t = [s_1, s_2, \dots, s_i, \dots, s_m]$ and represents the state of all machines. The initial state is $S_0 = [3, 1, \dots, 1]$, as all output buffers are empty and all machines are idle except for the first machine, which has started to process the first job (loaded automatically from IB_1). The final state is $S_{\check{T}} = [1, 1 \dots 3]$, where $\check{T}$ is the last transition epoch, which occurs when all jobs have completed processing and are placed in the final output buffer OB_m. We define S as the overall state space, such that $S_t \in S$. To overcome the problem where the state of system size grows exponentially in the number of robots as described in [67], the system state definition does not include: IB_i - since machine and output buffer status provide enough information on whether the robot should consider serving the machine, as well as the robot's location.

7.1.2 Action space

The action space A is a set of $m - 1$ actions $A = [a_1, a_2 \dots a_i \dots a_{m-1}]$ where action a_i is defined as a robot job transfer from M_i's OB_i to the input buffer IB_{i+1} of the next machine M_{i+1} in the processing sequence. The action space A is identical for both R_1 and R_2.

[4]The analysis in this chapter has been accepted to IEEE SMC 2013 Conference [7], and was recently submitted to the International Journal of Production Research. The first *RL* implementation for two robots was presented at the 20th International Conference on Production Research [6].

System transition times occur whenever one of the robots has just deposited a job at one of the machine's input buffers. Thus, τ (eq. 67) is used for $R_1$and R_2, and denoted as τ_1 and τ_2, respectively. Assuming the system starts at time 0, the system transition times can be defined as $t^1, t^2, ..., t^v, t^{v+1}$ with the system dynamics defined by $t^{v+1} = \min\{\tau_1, \tau_2\}$ for all $v = 0, 1, 2...$ At each of these times the system enters a new state S.

Each job must be transferred through $m-1$ machines (unloading from machine M_m is automatic) and so the minimum number of transitions (and actions) from the initial state to the final state will be $n(m-1)$. More than $n(m-1)$ transitions (actions) are possible if the state-action policy is non-optimal.

7.1.3 Rewards

The rewards, denoted as r_{t1i} for R_1 and r_{t2i} for R_2, are calculated at each transition time t for the robot that has just deposited a job and are defined as follows:

$$r_{t1i} = RAT_{ti} - JCT_{ti}; \text{ and} \tag{75}$$

$$r_{t2i} = JCT_{ti} - RAT_{ti}, \tag{76}$$

where r_{t1i} (r_{t2i}) is the reward achieved at time t by robot R_1 (R_2), if it serves M_i. RAT_{ti} is robot's arrival time from current location to machine M_i and JCT_{ti} is minimum job completion time in machine M_i (a job that has not yet been transferred to M_{i+1}). At each transition time t for the robot that has just deposited a job, reward is calculated for each machine M_i that the robot can serve, i.e., calculate r_{t1i} / r_{t2i} for $i = 1, 2 \dots m-1$ and when state of M_i is $s_i \neq 1$.

A penalty is defined for R_1, if the robot will have idle time when waiting near M_i for completion of J_j; and for R_2 when jobs have to wait on a machine until the robot arrives. For R_1, r_{t1i} is defined as follows: for each $i = 1 \dots m-1$ and $s_i \neq 1$, calculate $RAT_{ti} - JCT_{ti}$. If all values of $RAT_{ti} - JCT_{ti}$ are negative then R_1 chooses the maximum of $RAT_{ti} - JCT_{ti}$, i.e., R_1 serves M_i, which results in minimum robot idle time. Otherwise, at least one value of $RAT_{ti} - JCT_{ti}$ is positive and then the robot chooses the maximum of $RAT_{ti} - JCT_{ti}$ among positive values, i.e., R_1 serves M_i, which results in no robot idle time. For R_2, r_{t2i} is defined as follows: for each $i = 1 \dots m-1$ and $s_i \neq 1$, calculate $JCT_{ti} - RAT_{ti}$. If all values of $JCT_{ti} - RAT_{ti}$ are negative, then R_2 chooses the maximum of $JCT_{ti} - RAT_{ti}$, i.e., R_2 serves M_i, which results in the minimum waiting time of job J_j in OB_i. Otherwise, at least one value of $JCT_{ti} - RAT_{ti}$ is positive and then the robot chooses the maximum of $JCT_{ti} - RAT_{ti}$ among positive values, i.e., R_2 serves M_i, which results in no waiting time of job J_j in OB_i. Rewards and penalties contribute to reduction of the makespan value, i.e., $C_{\max}$.

A scheduling policy is obtained using Q-learning. The value of the system at state s_t, when selecting action a_t, is $Q(s_t, a_t)$ and is updated according to the Q-learning eq. (70). Since the non-stationarity of the multi-agent learning problem arises because all the agents in the system are learning simultaneously [67], our research uses a high value for $\bar{\alpha}$ and a low value for $\bar{\gamma}$ in order to give more weight to r_{t+1} and less weight for $Q(s_{t+1}, a_t)$. The $Q(s_t, a_t)$ value equation is updated for each transition $t = t^1, t^2, ..., t^v, t^{v+1}$.

As pointed out in Gil *et al.* [29], it is very hard within an episode to estimate the final performance of the schedule, i.e., the value of $C_{\max}$. Thus, in order to reinforce good schedules, Q-values must be updated at the end of a good schedule, i.e., at the end of one learning episode. In this research, a schedule is considered to be a good schedule if it results in $C_{\max} < C_{ave}$ (for calculation of C_{ave} see eq. (77)). Gil *et al.* [29], at the end of a good schedule, reinforce it by multiplying the Q-values by another variable. In *RR RL*, good schedules are reinforced by adding a predefined constant B, so $Q(s_t, a_t) = Q(s_t, a_t) + B$, since rewards can be negative and that may result in negative Q-values (see eqs. (75), (76), and (70)). Note that if there is a good schedule with negative Q-values then multiplying it by another value will not reinforce a good schedule, while adding B will reinforce it. Furthermore, *RR RL* use a constant B and not a variable, since a good schedule should be reinforced equally in the exploration (reinforcement of good schedules at the beginning of learning session may accelerate the learning process and so convergence to solution) and exploitation phases. The value of B was determined by prior simulations on small-size problems where optimal Q-values can be easily derived.

The transitions are terminated for the current episode when the system reaches its final state $S_{\check{T}} = [1, 1 \dots 3]$, which occurs when all jobs have completed processing and the robots have finished their work. If this event does not occur after a total of $\acute{E}$ learning episodes the process is terminated. Performance of robots, denoted as $C_{\max}(\acute{E}_i)$, is defined as the time it takes both robots to complete all job transfers.

At each transition epoch t, when robot R_1 or R_2 deposits a job at one of the machine input buffers it has to decide which action a_t to carry out next. The action a_t, is determined by a modified ε method, where ε is decreased over time (see eq. (69)) to encourage exploration at the beginning and exploitation as the robots improve. Here ε is the probability of selecting a machine at random ($0 \leq \varepsilon \leq 1$), that is a machine M_i $i = 1 \dots m-1$ is selected uniformly to be serviced by the "ready" robot. Otherwise, with probability $1 - \varepsilon$, the robot behaves greedily and selects an action according to the Q-values.

7.2 *RL* formulation – *RR* collaboration

The collaborative robot learning process uses a modified version of Q-learning. The robots operate simultaneously on parallel tracks and serve m machines. Four levels of collaboration (*Full*, *Pull*, *Push*, *None*) are defined based on *information sharing*, *learning*, and *assessment.*

Information sharing - Complete or None. Complete allows each robot to have complete knowledge about the current location of the other robot and the other robot's next location. None implies that the robots have no information.

Learning - Self or Joint. Self-learning indicates that each robot has direct influence on its Q-values with its own reward function with no interference by the other robot, while Joint learning implies that each robot has direct interference on the other robot's Q-values by using the other robot's reward function.

Assessment - The robots are equipped with the ability to evaluate their common performance. Performance C_{ave} is measured at the end of episode $\acute{E}_i$, calculating the average makespan over a span of X learning episodes starting from episode $\tilde{I}$ (eq. 77).

$$C_{ave} = \sum_{i=1}^{i=X} C_{\max}(\acute{E}_i)/\ X,\ \text{for } X = \tilde{I}, ..., \acute{E}, \tag{77}$$

where $C_{\max}(\acute{E}_i)$ is the makespan value of a problem that was achieved at the end of learning episode i. The measure, C_{ave}, enables the robots to decide when to collaborate by comparing C_{ave} to an adaptive threshold $\hat{C}_{\max}$, which represents the minimum average completion time achieved so far from the beginning of the learning trial. Note that starting from episode $\tilde{I}$, at each episode E_X $X = \tilde{I}, \ldots, \acute{E}$, C_{ave} is determined over the most recent X episodes.

The decision, if and when the robots should collaborate with one another, is determined on-line according to the following rules: (1) set an initial threshold $\hat{C}_{\max} = C_{\max}(\acute{E}_1)$ at the end of the first learning episode; (2) after X iterations compute the average learning performance measure C_{ave} (by eq. (77)); and (3) if $C_{ave} > \hat{C}_{\max}$ collaborate with the other robot (as defined in current collaboration level); otherwise, if $C_{ave} \leq \hat{C}_{\max}$ continue in autonomous mode and set $\hat{C}_{\max} = C_{ave}$.

Four levels of collaboration were defined and in each level both robots are trying to achieve the same goal, i.e., to minimize the makespan value $C_{\max}$.

Full - Robots work together with no "competition" where each robot, at each action selection, may advise the other robot about the next action. This depends on C_{ave} and $\hat{C}_{\max}$. Each robot has the ability to accept or reject the advice of which machine to serve next. The robots perform both self and joint learning and share full information by informing each other of their current and next location.

Pull - Only one robot can decide when and if to learn from the other robot and ask for advice (i.e., action). This depends on the previous performance of both robots, i.e., on C_{ave} and $\hat{C}_{\max}$. The second robot works and learns independently with no interference from the other robot. The robots share full information.

Push - One robot, say R_1, may force the other robot R_2 when and how to learn from it, by giving advice on which action to take. R_1 learns by itself, while R_2 is forced to accept the advice (i.e., action) of R_1, depending on previous performance of both robots, i.e., on C_{ave} and $\hat{C}_{\max}$. R_1 has full control on actions being performed. Robots share full information.

None - There is a "competition" between the autonomous robots; the robots do not share information and learn independently. There is no collaboration between the robots, i.e., they are two separate entities trying to achieve the same goal.

7.3 *RR* collaboration experiments

Collaboration algorithms for the four levels of our *RR RL* were programmed in *Matlab* (pseudo-code can be found in subsection (Appendix) 9.6). In order to evaluate the performance of each of the four levels of the *RR RL* collaboration algorithm (*Full*, *Pull*, *Push*, and *None*) we used an *UB error* as described in eq. (74). Furthermore, in order to determine the performance of two robots (same or of different type) and for each level of collaboration on given set of problems, we define the following performance measure:

$$\mathit{LB\ Fit} = 1 - \mathit{UB\ error}. \tag{78}$$

High *LB Fit* indicates that by implementing a specific level of collaboration, using robots of same or of different type, will result in high performance on a given set of problems.

Similar to the experiments conducted in section 6.3, our *RR RL* collaborative algorithm was also tested first for an *IM* problem (i.e., $p_{ij} = p$), and then for the more difficult problem with *NIM* (i.e., $p_{ij} = p_i$).

The following subsection presents and describes the simulation setup (subsection 7.3.1) and results (subsection 7.3.2) for the multi-robot scheduling problem formulated as a collaborative *RR RL* problem.

7.3.1 Simulation Setup

Evaluation tests for each level of collaboration and for identical robot types (*IRT*s) and non-identical robots types (*NIRT*s) for each variation, *IM* and *NIM* of the problem, were conducted. The number of robots, machines, and jobs were fixed at $k = 2$, $m = 7$, and $n = 10$, respectively. Processing times and robot job transfer times were sampled from uniform distributions. Job processing times were sampled uniformly between $[150, 250]$ or $[200, 300]$ or $[250, 350]$, where each defines a different set of problems. For *IM*s the processing time is sampled once, while for *NIM*s m samples are used, one for each machine M_i. A simulation is defined by a number of trials, where each trial consists of $\acute{E}$ learning episodes. Each simulation consists of 10 trials, and for each trial the total number of learning episodes is $\acute{E} = 100$ (this was empirically set based on a-priori analysis which indicated convergence after approximately 90 learning episodes), where after each sliding window of $\tilde{I} = 25$ learning episodes C_{ave} is calculated by eq. (77), and $B = 10$. Three types of robots (i.e., $Q = 3$, $q = 1, 2, 3$) were defined according to transfer times: (1) $q = 1$ fast robot - t_{i1} are sampled uniformly between $[5, 15]$, (2) $q = 2$ medium robot - t_{i2} are sampled uniformly between $[15, 25]$, and (3) $q = 3$ slow robot - t_{i3} are sampled uniformly between $[35, 45]$. These times are relative to robot empty travel times, which, without loss of generality, were defined as $te_{iq} = 1$ for $q = 1, 2, 3$. The robot empty times are taken as the same for each robot and associated with movements (without a job) between any two machines. Furthermore, three combinations of two robots were defined: two fast robots - R_1 and R_2 are type $q = 1$ noted as $(1, 1)$, fast and medium robots - R_1 type $q = 1$ and R_2 type $q = 2$ noted as $(1, 2)$, fast and slow robots - R_1 type $q = 1$ and R_2 type $q = 3$ noted as $(1, 3)$. For *IRT*s (i.e., $(1, 1)$), $m - 1$ samples for transferring times were sampled uniformly twice between $[5, 15]$ once for each robot, while for *NIRT*s (i.e., $(1, 2)$ and $(1, 3)$), $m - 1$ samples for transferring times were sampled uniformly twice, once for fast robot between $[5, 15]$ and once for medium robot between $[15, 25]$ or for slow robot between $[35, 45]$.

The following values were used for the reinforcement learning trials: $\bar{\beta} = 0.5$ in order to encourage exploration, and due to the non-stationary environment ([67]) a learning rate $\bar{\alpha} = 0.9$ and discount rate $\bar{\gamma} = 0.1$ were used.

For each learning episode, i.e., a single problem, the $\max\{LB\} = LB$ (see eq. 5) was calculated using $t_i = \min\{t_{iq}\}$ for $i = 1, ..., m - 1$, and with $\min\{UB\}$, *UB error* (74) and so *LB Fit* (78) were computed. The results are presented in terms of the average *UB error*.

7.3.2 Results and Discussion

This section presents average *UB error* results for *IM* (Table 7) and *NIM* (Table 8) for the robotic flow-shop scheduling problem. The best results for both, *IM* and *NIM*, were obtained by the *Full* collaboration mode while the worst results were obtained by the *None* collaboration mode. Furthermore, it can be seen that in some cases *Push* was better than *Pull* collaboration and in other cases the opposite. The average of the *UB error* decreases as processing times increase in all collaboration modes and in all robot type combinations and for both *IM* and *NIM*. Furthermore, as expected, the average *UB error* also decreases as robots become faster, i.e., $(1, 1)$ achieved best solutions, followed by $(1, 2)$, and the worst by $(1, 3)$.

Table 7 indicates the average *UB error*s for each of the cases, with *IM*.

	Processing Times	**Level of Collaboration**			
		Full	*Pull*	*Push*	*None*
(1, 1) *Fast, Fast*	p ~U[150, 250]	0.0048	0.0057	0.0077	0.046
	p ~U[200, 300]	0.004	0.0041	0.0043	0.04
	p ~U[250, 350]	0.0029	0.0029	0.0031	0.025
(1, 2) *Fast, Medium*	p ~U[150, 250]	0.02	0.038	0.0213	0.057
	p ~U[200, 300]	0.0165	0.033	0.017	0.052
	p ~U[250, 350]	0.0145	0.022	0.015	0.047
(1, 3) *Fast, Slow*	p ~U[150, 250]	0.032	0.043	0.045	0.083
	p ~U[200, 300]	0.03	0.038	0.038	0.065
	p ~U[250, 350]	0.028	0.034	0.036	0.059

Table 7: k=2 robot and IM.

Comparisons between average *UB error* of *Full*, *Pull*, *Push*, and *None* were conducted by performing Paired t-test at confidence level of 99%. It was found that between *Full* and *None* there was a statistically significant difference, while between *Pull* and *Push* there was no statistically significant difference. Between *Full* and *Pull*, *Full* and *Push* there was a statistically significant difference at a confidence level of 93%. Furthermore, when comparing average *UB error* by processing times (i.e., p~$U[150, 250]$, p~$U[200, 300]$, and p~$U[250, 350]$) it was found that there was a statistically significant difference.

In conclusion, *Full* resulted in *LB Fit* of $(1, 1)$ in the range of $99.52\% - 99.71\%$, $(1, 2)$ is $98\% - 98.55\%$, and $(1, 3)$ is $96.8\% - 97.2\%$. On the other hand, *None* resulting *LB Fit* of $(1, 1)$ is in the range of $95.4\% - 97.5\%$, of $(1, 2)$ is $94.3\% - 95.3\%$, and $(1, 3)$ is $91.7\% - 94.1\%$.

Table 8 indicates the average *UB error*s for each of the cases, with *NIM*.

Also, for the results in Table 8, comparisons between average *UB error* of *Full*, *Pull*, *Push*, and *None* were conducted by performing Paired t-test at a confidence level of 99%. As in *IM*, here for the *NIM* case, there was a statistically significant difference between average *UB error* of *Full* and *None* and between processing times (i.e., p~$U[150, 250]$, p~$U[200, 300]$, and p~$U[250, 350]$), while between the *Pull* and *Push* the results were not statistically different. Also, here (for *NIM*) there was a statistically significant difference between *Full* and *Pull*, *Full* and *Push* at a confidence level of 96%.

Robots	**Processing Times**	**Level of Collaboration**			
		Full	*Pull*	*Push*	*None*
(1, 1) *Fast, Fast*	$p \,\tilde{}\, U[150, 250]$	0.0019	0.0022	0.0019	0.0162
	$p \,\tilde{}\, U[200, 300]$	0.0017	0.0017	0.0017	0.0129
	$p \,\tilde{}\, U[250, 350]$	0.0008	0.0016	0.00088	0.0088
(1, 2) *Fast, Medium*	$p \,\tilde{}\, U[150, 250]$	0.0076	0.019	0.008	0.0219
	$p \,\tilde{}\, U[200, 300]$	0.0043	0.013	0.0044	0.0149
	$p \,\tilde{}\, U[250, 350]$	0.0036	0.006	0.0037	0.0103
(1, 3) *Fast, Slow*	$p \,\tilde{}\, U[150, 250]$	0.0196	0.0204	0.02	0.024
	$p \,\tilde{}\, U[200, 300]$	0.014	0.0146	0.0142	0.021
	$p \,\tilde{}\, U[250, 350]$	0.0072	0.009	0.0084	0.017

Table 8: k=2 robot and NIM.

7.4 Summary

In conclusion, *Full* resulted *LB Fit* of $(1,1)$ is in the range of 99.81% – 99.92%, of $(1,2)$ is 98.64% – 99.24%, and $(1,3)$ is 98.04% – 99.28%. On the other hand, *None* resulted *LB Fit* of $(1,1)$ is in the range of 98.38% – 99.12%, of $(1,2)$ is 97.81% – 98.97%, and $(1,3)$ is 97.6% – 98.3%. In most cases, *NIM* results in better performance than *IM*, i.e., most of the average *UB errors* in *NIM* are smaller. To evaluate differences between *IM* and *NIM*, a Paired t-test at a confidence level of 99% revealed that there is a significant difference between results for *IM* and *NIM*.

Section 7 presents a four-level collaborative *RR RL* algorithm that provides a schedule for two *IRT*s and *NIRT*s for both *IM* and *NIM*. The algorithms are considered to be generic, and can be used for various robot job transfer times (fast, medium, and slow) and processing times of machines than those presented in this section.

Through *UB error*, one will be able to select robots (fast, medium, or slow) and schedule them according to system characteristics and factory constraints. For example, assume that following criteria exist: (a) cost of each robot - determined according to transfer and empty return times, i.e., fast robot is expensive, medium robot is medium, and slow robot is cheap. (b) For each time that the factory is late in completing an order of n completed parts, i.e., $C_{\max}$ is bigger than optimal $C_{\max}$, there is a fine of ($C_{\max}$ – (optimal $C_{\max}$)), and (c) each level of collaboration consumes different time and requires different hardware, i.e., implementation of *Full* in the most expensive while *None* is the cheapest. It can be seen that according to each weight given to each criterion one will be able to decide which robots to select and which level of collaboration to use in order to schedule them. Thus, if the fine ($C_{\max}$ – (optimal $C_{\max}$)) is enormous compared to the costs of robots and to costs of the level of collaboration, then it is best to select two fast robots that implement *Full* level of collaboration.

8 Summary and Future Research

We studied three different robotic flow-shop scheduling problems, where in each of the problems a set of n identical jobs has to be scheduled on a set of m machines, and the scheduling objective is to minimize the makespan. Moreover, in all problems robots are responsible for transporting jobs between successive machines. In the first problem (*Problem 1*) we study the case where there is a single robot, and unlimited buffer between the machines. In the second problem (*Problem 2*), we consider the case where the processing times are both machine and job independent, the robots move on the same track, and there are no-idle and no-wait restrictions. We consider the case where the scheduler has to select a set of robots (from a predefined set of robot types) and assign each robot to a portion of the track. Once the robots are selected and assigned, the scheduler has to schedule the robots as well. In this problem, in addition to minimizing the makespan, the scheduler aims to minimize the total robot selection cost. In the third problem (*Problem 3*), there are two robots that are moving on parallel tracks.

Our main results for each of the above three problems are listed below:

- For *Problem 1*, we provide a polynomial time procedure for solving the special case that includes three machines. The procedure is based on decomposing the problem into a set of subproblems and providing an optimal schedule for each of the subproblems. Moreover, for more than three machines, we design a unique collaborative *RL*-based procedure to solve the problem. In our collaborative *RL* procedure the robot collaborates with an adviser (agent), and has the following three robot cognitive abilities: (*i*) the ability to assess the quality of its own decisions; (*ii*) a short-term assessment ability for evaluating the quality of an advice; and (*iii*) a long-term assessment for evaluating the adviser's skill levels. Thus, the robot in the short term was able to accept or reject advice, and in the long run was able to decide whether to collaborate or discard the adviser. For two types of agents (i.e., agent with fixed level of expertise and agent with fatigue), we show that the robot was able to achieve better results and collaborated more (asking for more advise) with an expert rather than with a novice.

- For *Problem 2*, as there are two criteria (minimizing the makespan and minimizing the total robot selection cost), we define four variations of the problem. The first is to find a schedule that minimizes the sum of the two conflicting criteria; the second and the third are to minimize one criterion, given an upper bound on the value of the other criterion; while the last variation consists of finding the entire set of Pareto-optimal points with respect to the two criteria. We show that the first problem variation is solvable in polynomial time while the other three variations are $\mathcal{NP}$-hard. For these $\mathcal{NP}$-hard problems, we show that a pseudo-polynomial time algorithm and a fully polynomial approximation scheme (*FPTAS*) exists. Moreover, we derive three important special cases that are solvable in polynomial time.

- For solving *Problem 3*, we design another unique collaborative *RL*-based procedure, where robots are able to communicate in one of the following levels of collaboration: (*i*) *Full*, (*ii*) *Push*, (*iii*) *Pull*, and (*iv*) *None*. Each level was defined according to three characteristics: information sharing (complete or none), learning (self and joint), and assessment (joint). By applying an extensive experimental study, we were able to obtain the following results: (*i*) using the *Full* level of collaboration provides the best results for different types of robots, while *None* leads to the worst quality solutions. (*ii*) There is no difference in the quality of the results between *Pull* and *Push* (with level of confidence of 99%). It is important to note that despite the fact that the environment becomes non-stationary, our *RL*-based procedure was able to produce a makespan value that is at most 1.03 times bigger than the value of a tight lower bound, when using a full collaboration between the robots.

In future research, the complexity of *Problem 1* for any arbitrary number of machines should be classified. Future research may also include the analysis of *Problem 2,* for the more general case where job processing times may be machine-dependent, or for the case where the working space of each robot is not partitioned, so the limitation on the movements of each robot is based on the position of the robot from its left and right side, in order to avoid collisions. The most important challenge for future research is to identify the exact complexity status of *Problem 3.* With regard to *RL*, the *RL* collaborative algorithms should be expanded for problems where jobs are non-identical and thus job schedule decisions have to be provided as well. Furthermore, it is important to analyze differences between two performance measures of the robot(s) that were used in *RA* and *RR*, i.e., the performance measure used in *RA* (eq. (71)) calculated C_{ave} according to X last episodes, while the performance measure used in *RR* (eq. (77)) calculated C_{ave} according to X learning episodes starting from the first episode until episode X. In addition, for both performance measures, more challenges arise: determine the best value of X for a given number of learning episodes E, and through that find the dependency between X and E.

For the single robot system, experiments should be expanded to test the quality of *RA RL* in a real human-robot system where a robot is required to control the system behavior on-line. For the multi-robot system, algorithms in which all robots can autonomously determine how and when to switch collaboration levels should be developed.

References

[1] Adiri, I., and Pohoryles, D., 1982, Flow-Shop/No-Idle or No-Wait Scheduling to Minimize the Sum of Completion Times, *Naval Research Logistics*, **29**, 495–504.

[2] Adiri, I., and Amit, N., 1984, Openshop and Flowshop Scheduling to Minimize Sum of Completion Times, *Computers & Operations Research*, **11** (3), 275–284.

[3] Ageev, A.A., and Baburin, A.E., 2007, Approximation Algorithms for UET Scheduling Problems with Exact Delays, *Operation Research Letters*, **35** (4), 533–540.

[4] Agnetis, A., 2000, Scheduling No-Wait Robotic Cells with Two and Three Machines, *European Journal of Operational Research*, **123** (2), 303–314.

[5] Agnetis, A., and Pacciarelli, D., 2000, Part Sequencing in Three-Machine No-Wait Robotic Cells, *Operations Research Letters*, **27** (4), 185–192.

[6] Arviv K., Stern H., and Edan Y., 2009, Flow-Shop Problem with Job Transfer Robots Using Reinforcement Learning, *20th International Conference on Production Research*, China.

[7] Arviv K., Stern H., and Edan Y., 2013, Collaborative Reinforcement Learning for a Two Robot Job Transfer Flow-shop Scheduling Probelm, *IEEE SMC 2013 Conference*, UK.

[8] Aydin, M.E., and Oztemel, E., 1999, Dynamic Job-Shop Scheduling Using Reinforcement Learning Agents, *Robotics and Automation Systems*, **33**, 16-178.

[9] Baxter, J., Tridgell, A., and Weaver, L., 1998, TDLeaf(λ): Combining Temporal Difference Learning with Game-Tree Search, *Journal of Intelligent Information Processing Systems*, **5**, 39-43.

[10] Bhanu, B., Leang, P., Cowden, C., Lin, Y., and Patterson M., 2001, Real-Time Robot Learning, *IEEE International Conference on Robotics and Automation,* **1**, 491-498.

[11] Brucker, P., and Shakhlevich, N.V., 2011, A Polynomial-Time Algorithm for a Flow-Shop Batching Problem with Equal-Length Operations, *Journal of Scheduling*, **14** (4), 371-389.

[12] Caroline, C., and Craig, B., 1998, The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems, in Proceedings AAAI of the *Fifteenth National Conference on Artificial Intelligence*, 746–752.

[13] Che, A., Chu, C., and Chu, F., 2002, Multi Cyclic Hoist Scheduling with Constant Processing Times, *IEEE Transactions on Robotics and Automation*, **18** (1), 69–80.

[14] Che, A., Chu, C., and Levner, E., 2003, A Polynomial Algorithm for 2-Degree Cyclic Robot Scheduling, *European Journal of Operational Research*, **38**, 1275-1285.

[15] Che, A., and Chu, C., 2005, Multi-Degree Cyclic Scheduling of Two Robots in a No-Wait Flowshop, *IEEE Transactions on Automation Science and Engineering*, **2** (2), 173–183.

[16] Che, A., and Chu, C., 2008, Optimal Scheduling of Material Handling Devices in a PCB Production Line Problem Formulation and a Polynomial Algorithm, *Mathematical Problems in Engineering*, Article ID **364279**, 21 Pages.

[17] Che, A., and Chu, C., 2009, Multi-Degree Cyclic Scheduling of a No-Wait Robotic Cell with Multiple Robots, *European Journal of Operational Research*, **199**, 77-88.

[18] Che, A., Hu, H., Chabrol, M., and Gourgand, M., 2011, A Polynomial Algorithm for Multi-Robot 2-Cyclic Scheduling in a No-Wait Robotic Cell, *Computers and Operations Research*, **38**, 1275-1285.

[19] Dahl, T.S., Matarić, M.J., and Sukhatme, G.S., Scheduling with Group Dynamics: a Multi-Robot Task Allocation Algorithm Based on Vacancy Chains, *Technical Report*, (*http://cres.usc.edu/Research/files/81.pdf*)

[20] Choi, Y.C., and Ahn, H.S., 2010, A Survey on Multi-Agent Reinforcement Learning: Coordination Problems, *IEEE/ASME International Conference on Mechatronics and Embedded Systems and Applications*, 81-86.

[21] Chu, C., 2006, A Faster Polynomial Algorithm for 2-Cyclic Robotic Scheduling, *Journal of Scheduling*, **9** (5), 453–68.

[22] Dell'Amico, M., 1996, Shop Problems with Two-Machines and Time Lags, *Operations Research*, **44,** 777-787.

[23] Erfu, Y., and Dongbing, G., 2006, Multiagent Reinforcement Learning for Multi-Robot Systems: A Survey, *9th International Conference on Control, Automation, Robotics and Vision*, 1-6.

[24] Ergun, F., Sinha, R., and Zhang, L., 2002, An Improved FPTAS for the Restricted Shortest Path Problem, *Information Processing Letters*, **83**, 287-291.

[25] Franklin, A.J., Mitchell, T.M., and Thrun, S., 1996, Recent Advances in Robot Learning (Introduction), *Computer Science Department Stanford University, Machine Learning,* **23**, 117-119.

[26] Gabel, T., and Riedmiller, M., 2011, Distributed Policy Search Reinforcement Learning for Job-Shop Scheduling Tasks, *International Journal of Production Research*, **50**, 41-61.

[27] Garey, M.R., Johnson, D.S., and Sethi, R., 1976, The Complexity of Flowshop and Jobshop Scheduling, *Mathematics of Operations Research*, **1**, 117-129.

[28] Garroppo, R.G., Giordano, S., and Tavanti, L., 2010, A Survey on Multi-Constrained Optimal Path Computation: Exact and Approximate Algorithms, *Computer Networks*, **54**, 3081-3107.

[29] Gil, A., Stern, H., Edan, Y., and Kartoun, U., 2008, A Scheduling Reinforcement Learning Algorithm, *Proceedings of the ASME International Symposium on Flexible Automation*, Atlanta, Georgia, USA.

[30] Gil, A., Stern, H., and Edan, Y., 2009, A Cognitive Robot Collaborative Reinforcement Learning Algorithm, *International Conference Modeling and Simulation,* May 25-29, **53**, 430-433.

[31] Glorennec, P.Y., 2000, Reinforcement Learning: an Overview, *European Symposium on Intelligent Thechniques, Aachen, Germany*, 17-35.

[32] Goncharov, Y., and Sevastyanov, S., 2009, The Flow Shop Problem with No-Idle Constraints: a Review and Approximation, *European Journal of Operational Research,* **196**, 450–456.

[33] Graham, R.L., Lawler, E.L., Lenstra, J.K., and Rinnooy Kan, A.H.G., 1979, Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey, *Annals of Discrete Mathematics*, **3**, 287-326.

[34] Guo, M., Liu, Y., and Malec, J., 2004, A New Q-Learning Algorithm Based on the Metropolis Criterion, *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, **34**, 2140-2143.

[35] Gerhard, W., 1999, Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. *Cambridge, MA: MIT Press.*

[36] Hall, N.G., and Sriskandarajah, C., 1996, A Survey of Machine Scheduling Problems with Blocking and No-Wait Process, *Operations Research*, **44**, 510–525.

[37] Hassin, R., 1992, Approximation Schemes for the Restricted Shortest Path Problem, *Mathematics of Operations Research*, **17**, 36-42.

[38] Hurink, J., and Knust, S., 2001, Makespan Minimization for Flow-Shop Problems with Transportation Times and a Single Robot, *Discrete Applied Mathematics*, **112**, 199-216.

[39] Johnson, S.M., 1954, Optimal Two and Three-Stage Production Schedules with Setup Times Included, *Naval Research Logistics,* **1**, 61–68.

[40] Kalczynski, P.J., and Kamburowski, J., 2007, On No-Wait and No-Idle Flow Shops with Makespan Criterion, *European Journal of Operational Research*, **178** (3), 677-685.

[41] Kartoun, U., Stern, H., and Edan, Y., 2010, Human-Robot Collaborative Reinforcement Learning Algorithm, *Journal of Intelligent Robotic Systems,* **60**, 217-239.

[42] Karuno, Y., and Nagamochi, H., 2003, A Better Approximation for the Two-Machine Flowshop Scheduling Problem with Time Lags, *Lecture Notes in Computer Science*, **2906**, 309-318.

[43] Karzanov, A.V., and Livshits, E.M., 1978, Minimal Quantity of Operators for Serving a Homogeneous Linear Technological Process, *Automation and Remote Control*, **3**, 445–450.

[44] Kats, V., and Levner, E., 1997, Minimizing the Number of Robots to Meet a Given Cyclic Schedule, *Annals of Operations Research*, **69**, 209-226.

[45] Kats, V., Levner, E., and Meyzin, L., 1999, Multiple-Part Cyclic Hoist Scheduling Using a Sieve Method, *IEEE Transactions on Robotics and Automation*, **15** (4), 704–13.

[46] Kats, V., and Levner, E., 2002, Cyclic Scheduling in a Robotic Production Line, *Journal of Scheduling*, **5** (1), 23–41.

[47] Kemmerich, T., and Buning, H.K., 2011, Coordination in Large Multiagent Reinforcement Learning Problems, *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT),* 236-239.

[48] Kenneth, R.B., 1974, Introduction to Sequencing and Scheduling, *USA: John Wiley & Sons Inc.*, Sections **1-3**, 2-176 & **6-7** 133-191.

[49] Kise, H., 1991, On an Automated Two-Machine Flow-Shop Scheduling Problem with Infinite Buffer, *Journal of the Operations Research Society of Japan*, **34**, 354-361.

[50] Kise, H., Shioyama, T., and Ibaraki, T., 1991, Automated Two-Machine Flowshop Scheduling: a Solvable Case, *IIE Transactions*, **23**, 10–16.

[51] Laetitia, M., Guillaume, J.L., and Nadine, L.F.P., 2007, Hysteretic Q-Learning: An Algorithm for Decentralized Reinforcement Learning in Cooperative Multi-Agent Teams, *IEEE/RSJ international conference on Intelligent Robots and Systems*, 64-69.

[52] Lamoudan, T., Khoukhi, F.E., Alaoui, A.E.H., and Boukachour, J., 2011, Flow-Shop Scheduling Problem with Transportation Times, Two Robots and Inputs/Outputs with Limited Capacity, *International Journal of Intelligent Computing Research*, **2**, 244-253.

[53] Lebacque, V., and Brauner, N., 2008, Flowshops with Material Handling: A Critical Comparison of Different Existing Models, *France Laboratoire Leibniz-IMAG*.

[54] Lee, C.Y., and Chen, Z.L., 2001, Machine Scheduling with Transportation Considerations, *Journal of Scheduling*, **4**, 3–24.

[55] Lee, C.Y., and Strusevich, V., 2005, Two-Machine Shop Scheduling with an Uncapacitated Interstage Transporter, *IIE Transactions*, **37** (8), 725-736.

[56] Lei, L., and Wang, T.J., 1994, Determining Optimal Cyclic Hoist Schedules in a Single-Hoist Electroplating Line, *IIE Transactions*, **26** (2), 25–33.

[57] Leung, J., and Zhang, G., 2003, Optimal Cyclic Scheduling for Printed-Circuit-Board Production Lines with Multiple Hoists and General Processing Sequences, *IEEE Transactions on Robotics and Automation*, **19** (3), 480–484.

[58] Leung, J., Zhang, G., Yang, X., Mak, R., and Lam, K., 2004, Optimal Cyclic Multi-Hoist Scheduling: A Mixed Integer Programming Approach, *Operations Research*, **52** (6), 965–976.

[59] Leung J., and Levner E., 2006, An Efficient Algorithm for Multi-Hoist Cyclic Scheduling with Fixed Processing Times, *Operation Research Letters*, **34 (4)**, 465–572.

[60] Levner, E., Kogan, V., and Levin, I., 1995, Scheduling a Two-Machine Robotic Cell: A Solvable Case, *Annals of Operations Research,* **57**, 217–232.

[61] Levner, E., Kats, V., and Levit, V.E., 1997, An Improved Algorithm for Cyclic Flowshop Scheduling in a Robotic Cell, *European Journal of Operational Research*, **97**, 500-508.

[62] Levner, E., Kats, V., De Pablo, D.A.L., and Cheng, T.C.E., 2011, Complexity of Cyclic Scheduling Problems: A State-of-the-Art Survey, *Computers and Industrial Engineering*, **59**, 352-361.

[63] Lim, A., Rodrigues, B., and Wang, C., 2006, Two-Machine Flow Shop Problems with a Single Server, *Journal of Scheduling*, **9**, 515–543.

[64] Ling, S., and Guang, C.X., 2011, Complexity Results for Flow-shop Scheduling Problems with Transportation Delays and a Single Robot, *Journal of Applied Mathematics & Bioinformatics*, **1** (1), 135-142.

[65] Liviu. P., and Sean, L., 2005, Cooperative Multi-Agent Learning: The State of the Art, *Autonomous Agents and Multi-Agent Systems*, **11**, 387–434.

[66] Lorenz, D., and Raz, D., 2001, A Simple Efficient Approximation Scheme for the Restricted Shortest Path Problem, *Operations Research Letters*, **28**, 213-219.

[67] Lucian, B., Robert, B., and Bart, D.S., 2008, A Comprehensive Survey of Multi Agent Reinforcement Learning, *IEEE Transactions On Systems, Man and Cybernetics, Part C: Cybernetics*, **38**, 156-172.

[68] Martinez, Y., Nowe, A., Suarez, J., and Bello, R., 2011, A Reinforcement Learning Approach for the Flexible Job-Shop Scheduling Problem, *Learning and Intelligent Optimization*, **6683**, 253-262.

[69] Mosheiov, G., Oron, D., and Ritov, Y., 2004, Flow-Shop Batch Scheduling with Identical Processing Time Jobs, *Naval Research Logistics*, **51**, 783–799.

[70] Mosheiov, G., and Oron, D., 2005, A Note on Flow-Shop and Job-Shop Batch Scheduling with Identical Processing Time Jobs, *European Journal of Operational Research*, **161**, 285–291.

[71] Ohshifa, T., Shin, J.S., Miyazaki, M., and Lee, H.H., 2008, A Cooperative Behavior Learning Control of Multi-Robot Using Trace Information, *Artif Life Robotics*, **13**, 144-147.

[72] Orlin, J.B., 1982, Minimizing the Number of Vehicles to Meet a Fixed Periodic Schedule: An Applications of Periodic Posets, *Operations Research*, **30**, 760–776.

[73] Panwalkar, S.S., 1991, Scheduling of a Two-Machine Flowshop with Travel Time Between Machines, *Journal of the Operations Research Society*, **42**, 609-613.

[74] Peng, J., and Williams, R., 1996, Incremental Multi-Step Q-Learning, *Machine Learning*, **22**, 283-290.

[75] Pereda, J., Martin-Ortiz, M., de Lop, J., and de la Paz, J., 2011, Study of a Multi-Robot Collaborative Task through Reinforcement Learning, *Foundations on Natural and Artificial Computation*, **6686**, 185-191.

[76] Peter, S., 2003, Flow-Shop Scheduling Based on Reinforcement Learning Algorithm, *Production Systems and Information Engineering*, **1**, 83-90.

[77] Phillips, L.W., and Unger, P.S., 1976, Mathematical Programming Solution of a Hoist Scheduling Program, *AIIE Transactions*, **8**, 219-225.

[78] Pinedo, M., 2001, *Scheduling: Theory, Algorithms and Systems, Second Ed*, Prentice-Hall, NJ.

[79] Ribeiro, C., 2002, Reinforcement Learning Agents, *Artificial Intelligence Review*, **17**, 223-250.

[80] Ruiz, R., Vallada, E., and Fernandez-Martinez, C., 2009, Scheduling in Flowshops with No-Idle Machines, *Computational Intelligence in Flow shop and Job Shop Scheduling in Computational Intelligence*, **230**, 21-51.

[81] Saadani, N.E.H., Guinet, A., and Moalla, M., 2003, Three Stage No-Idle Flow-Shops, *Computers and Industrial Engineering*, **44**, 425–434.

[82] Shabtay, D., Arviv, K., Stern, H., and Edan, Y., 2014, A Combined Robot Selection and Scheduling Problem for Flow-Shops with No-Wait Restrictions, *Omega*, **43**, 96-107.

[83] Siang, K.S., Kai, W.O., and Gerald, S., 2003, A Foundation for Robot Learning, *4th International Conference on Control and Automation*, 649-653.

[84] Stern, H., and Vitner, G., 1990, Scheduling Parts in a Combined Production-Transportation Work Cell, *Journal of the Operations Research Society*, **41**, 625-632.

[85] Stern, H., Arviv, K., and Edan, Y., 2011, Flow-Shop Robotic Scheduling with Collaborative Reinforcement Learning, *21th International Conference on Production Research*, Germany.

[86] Sutton, R.S., 1998, Learning to Predict by the Method of Temporal Differences, *Machine Learning,* **3**, 9-44.

[87] Sutton, R.S., and Barto, A.G.,1998, Reinforcement Learning: An Introduction. *Cambridge, MA: MIT Press.* Section **1**, 2-176.

[88] Tang, L., and Liu, P., 2009, Two-Machine Flowshop Scheduling Problems Involving a Batching Machine with Transportation or Deterioration Consideration, *Applied Mathematical Modelling*, **33** (2), 1187–1199.

[89] Watkins, C.J.C.H., 1989, Learning from Delayed Rewards, PhD Thesis, *Cambridge University, Cambridge, England.*

[90] Wei, Y., and Zhao, M., 2004, Composite Rules Selection using Reinforcement Learning for Dynamic Job-Shop Scheduling, *IEEE Conference on Robotics, Automation and Mechatronics*, **2**, 1083-1088.

[91] Wei, Y., and Zhao, M., 2005, A Reinforcement Learning Based Approach to Dynamic Job-shop Scheduling, *Acta Automatica Sinica*, **31**, 765-771.

[92] Yu, W., 1996, The Two-Machine Flow Shop Problem with Delays and the One-Machine Total Tardiness Problem, Ph.D. Thesis, *Technische Universiteit Eindhoven.*

[93] Yu, W., Hoogeveen, H., and Lenstra, J.K, 2004, Minimizing Makespan in a Two-Machine Flow Shop with Delays and Unit-Time Operations is $\mathcal{NP}$-hard, *Journal of Scheduling*, **7** (5), 333-348.

[94] Yusuke, T., and Taketoshi, Y., 1999, An Application of Reinforcement Learning to Manufacturing Scheduling Problems, *IEEE International Conference on Systems, Man and Cybernetics*, **4**, 534-539.

[95] Zhicong, Z., Weiping, W., Shouyan, Z., and Kaishun, H., 2013, Flow-shop Scheduling with Reinforcement Learning, *Asia-Pacific Journal of Operational Research,* **30**, 5.

[96] Zhu, W., 2001, Vision-based Reinforcement Learning for Robot Navigation, *International Joint Conference on Neural Networks*, **2**, 1025-1030.

9 Appendices

9.1 The analysis of *Case 1*

This subsection analyzes the case where M_1 is the bottleneck in the system. Thus, the condition in (6) for $f = 1$ holds and eq. (3) for $f = 1$ provides the lower bound on the makespan value. We further divide *Case 1* into two subcases. The first is where $p_1 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$, and the second is the opposite case. Below, we provide optimal schedules for each of the two subcases.

9.1.1 The analysis of subcase 1.1

We further divide subcase 1.1, where $p_1 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$, into three subcases. The first, subcase 1.1.1, is where $p_2 = \max\{p_2, p_3, t_2 + te_2\}$, the second subcase 1.1.2 is where $t_2 + te_2 = \max\{p_2, p_3, t_2 + te_2\}$, and the third subcase 1.1.3 is where $p_3 = \max\{p_2, p_3, t_2 + te_2\}$.

The analysis of subcase 1.1.1 For subcase 1.1.1, where $p_1 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ and $p_2 = \max\{p_2, p_3, t_2 + te_2\}$, we define the following schedule (Schedule 1.1.1):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S^m_{1j}, C^m_{1j}] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_j on M_2 during time interval $[S^m_{2j}, C^m_{2j}] = [jp_1 + t_1, jp_1 + t_1 + p_2]$ for $j = 1, ..., n$.
- Schedule job J_j on M_3 during time interval $[S^m_{3j}, C^m_{3j}] = [(j+1)p_1 + t_1 + t_2, (j+1)p_1 + t_1 + t_2 + p_3]$ for $j = 1, ..., n-1$.
- Schedule job J_n on M_3 during time interval $[S^m_{3n}, C^m_{3n}] = [np_1 + t_1 + p_2 + t_2, np_1 + t_1 + p_2 + t_2 + p_3]$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $[S^r_{1j}, C^r_{1j}] = [jp_1, jp_1 + t_1]$ for $j = 1, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $[S^r_{2j}, C^r_{2j}] = [(j+1)p_1 + t_1, (j+1)p_1 + t_1 + t_2]$ for $j = 1, ..., n-1$.
- Move job J_n from M_2 to M_3 during time interval $[S^r_{2n}, C^r_{2n}] = [np_1 + t_1 + p_2, np_1 + t_1 + p_2 + t_2]$.

- Move the robot empty from M_2 to M_1 during time interval $[p_1+t_1, p_1+t_1+te_1]$.
- Move the robot empty from M_3 to M_1 during time interval $[jp_1+t_1+t_2, jp_1+t_1+t_2+\sum_{i=1}^{2} te_i]$ for $j = 2, ..., n-1$.
- Move the robot empty from M_3 to M_2 during time interval $[np_1+t_1+t_2, np_1+t_1+t_2+te_2]$.

Schedule 1.1.1 is illustrated in Figure 8 below for $n = 4$ jobs with empty return moves shown in bold.

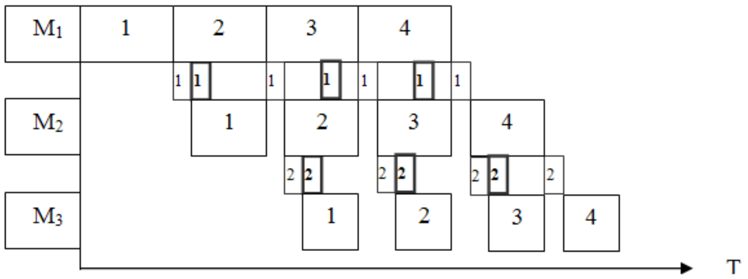

Figure 8: Schedule 1.1.1.

Lemma 8 *Schedule 1.1.1 is an optimal schedule for subcase 1.1.1.*

Proof. We start by showing that schedule 1.1.1 is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$; that $S^m_{2,j+1} = (j+1)p_1 + t_1 \geq C^m_{2j} = jp_1 + t_1 + p_2$; that $S^m_{3,j+1} = (j+2)p_1 + t_1 + t_2 \geq C^m_{3j} = (j+1)p_1 + t_1 + t_2 + p_3$ for $j = 1, ..., n-2$; and that $S^m_{3n} = np_1 + t_1 + p_2 + t_2 > C^m_{3,n-1} = np_1 + t_1 + t_2 + p_3$, where the three last inequalities follows from the fact that for *subcase 1.1.1* we have that $p_1 \geq p_2 \geq p_3$, implies that there is no overlap between processing operations on the same machine. Moreover, the fact that $C^m_{1j} = S^r_{1j} = jp_1$; that $C^r_{1j} = S^m_{2j} = jp_1 + t_1$; that $C^m_{2j} = jp_1 + t_1 + p_2 \leq S^r_{2j} = (j+1)p_1 + t_1$; that $C^r_{2j} = (j+1)p_1 + t_1 + t_2 = S^m_{3j}$ for $j = 1, ..., n-1$; and that $C^r_{2n} = np_1 + t_1 + p_2 + t_2 = S^m_{3n}$ implies that there is no overlap between processing and transferring operations of job J_j for $j = 1, \ldots, n$. Thus, Condition 1 holds.

Once job J_1 is completed on machine M_1, the robot moves this job to M_2 during time interval $[p_1, p_1+t_1]$ and then return empty to M_1 during time interval $[p_1+t_1, p_1+t_1+te_1]$. Then for $j = 2, ..., n-1$ the robot moves job J_j from M_1 to M_2 during time interval $[jp_1, jp_1+t_1]$; job J_{j-1} from M_2 to M_3 during time interval $[jp_1+t_1, jp_1+t_1+t_2]$, and return empty to M_1 during time interval $[jp_1+t_1+t_2, jp_1+t_1+t_2+\sum_{i=1}^{2} te_i]$. Then the robot moves job J_n from M_1 to M_2 during time interval $[np_1, np_1+t_1]$ and job J_{n-1} from M_2 to

M_3 during time interval $[np_1+t_1, np_1+t_1+t_2]$. Lastly, the robot return empty from M_3 to M_2 during time interval $[np_1+t_1+t_2, np_1+t_1+t_2+te_2]$ and moves job J_n from M_2 to M_3 during time interval $[np_1+t_1+p_2, np_1+t_1+p_2+t_2]$. The fact that $p_2 \geq t_2+te_2$ implies that there is no overlap between the two last operations. Thus, Condition 2 holds.

The fact that $C^m_{1j} = S^r_{1j} = jp_1$ implies that job J_j is ready to move from M_1 to M_2 at time S^r_{1j}. Moreover, the fact that $C^m_{2j} = jp_1+t_1+p_2 \leq S^r_{2j} = (j+1)p_1+t_1$ follows from the fact that $p_1 \geq p_2$ and implies that job J_j is ready to move from M_2 to M_3 at time S^r_{2j}. Thus, Condition 3 holds.

It is implied from the feasibility of schedule 1.1.1 that the completion time of job n on M_3 is at time $C^m_{3n} = np_1 + \sum_{i=1}^{2} t_i + p_2 + p_3$. The fact that this time matches lower bound value in eq. (9) implies that this schedule is optimal for Case 1.1.1. ■

The analysis of subcase 1.1.2 In subcase 1.1.2, where $p_1 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ and $t_2+te_2 = \max\{p_2, p_3, t_2+te_2\}$, we first update LB_1 and provide a revised (tighter) bound denoted by LB'_1. To do so, we consider two possible scenarios: (a) the robot does not perform two consecutive moves from M_1 to M_2, and (b) the robot makes at least once two consecutive moves from M_1 to M_2. Let us first consider case (a). In this case the robot moves each job from M_1 to M_2 and then to M_3 before moving to the next job, i.e., after transferring job J_j $(j=1,...,n)$ form M_1 to M_2, the robot waits beside M_2 for the completion of J_j on this machine and then move it to M_3. It thus implies that the robot will start move J_1 from M_1 to M_2 not earlier than p_1, and will start move J_j from M_1 to M_2 not earlier than $p_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$ for $j=2,...,n$. Thus, job J_n will not finish its processing on M_3 earlier than at time

$$LB'_1(a) = p_1+(n-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+t_1+p_2+t_2+p_3 \geq np_1+t_1+p_2+t_2+p_3 = LB_1.$$

Consider next case (b) where the robot makes at least once two consecutive moves from M_1 to M_2 (and thus also at least once two consecutive moves from M_2 to M_3). We further divide case (b) into two subcases. The first $(b1)$ where the *last* two consecutive moves from M_2 to M_3 are done on jobs J_{n-1} and J_n, and the second $(b2)$ when the *last* two consecutive moves from M_2 to M_3 are done on jobs J_{x-1} and J_x where $2 \leq x \leq n-1$. In subcase $(b1)$, jobs J_{n-1} and J_n are both in M_2 not earlier than at time np_1+t_1. Thus, the move of J_{n-1} from M_2 to M_3 cannot start before np_1+t_1, which further implies that move J_n from M_2 to M_3

cannot start before time $np_1 + t_1 + t_2 + te_2$. Thus, the makespan value is lower bounded by

$$LB_1'(b1) = np_1 + \sum_{i=1}^{2} t_i + te_2 + t_2 + p_3 \geq LB_1 = np_1 + \sum_{i=1}^{2} t_i + p_2 + p_3.$$

In subcase $(b2)$, the fact that the last two consecutive moves from M_2 to M_3 are done on jobs J_{x-1} and J_x, implies that job J_j for $j = x+1, \ldots, n$ is transferred from M_1 to M_2 and then from M_2 to M_3 (i.e., the robot waits beside M_2 for the completion of each job J_j for $j = x+1, \ldots, n$). The earliest time in which jobs J_{x-1} and J_x are both in M_2 is at $xp_1 + t_1$, and thus the move of J_{x-1} from M_2 to M_3 will not start before this time. This further implies that the move of J_x from M_2 to M_3 will not start before $xp_1 + t_1 + t_2 + te_2$, which also implies that the move of J_{x+1} from M_1 to M_2 will not start before $xp_1 + t_1 + t_2 + te_2 + t_2 + \sum_{i=1}^{2} te_i$. The fact that the robot waits beside M_2 for the completion of each job J_j for $j = x+1, \ldots, n$ further implies that the earliest time that the robot will be available for moving job J_n from M_1 to M_2 is

$$S_{1n}^r = xp_1 + \sum_{i=1}^{2} t_i + te_2 + t_2 + \sum_{i=1}^{2} te_i + (n-x-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2).$$

Thus, a lower bound for makespan value in this case is

$$\begin{aligned} LB_1'(b2) &= S_{1n}^r + \sum_{i=1}^{2} t_i + p_2 + p_3 = xp_1 + 2\sum_{i=1}^{2} t_i + te_2 + t_2 + \\ &\quad \sum_{i=1}^{2} te_i + (n-x-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) + p_2 + p_3. \end{aligned}$$

Since $p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i \geq p_1$ and $x \geq 2$, this value is lower bounded by

$$LB_1'''(b2) = (n-1)p_1 + 2\sum_{i=1}^{2} t_i + te_2 + t_2 + \sum_{i=1}^{2} te_i + p_2 + p_3 \geq np_1 + \sum_{i=1}^{2} t_i + te_2 + t_2 + p_3 = LB_1'''(b1).$$

To conclude our analysis, we have that if scenario (a) is selected, then

$$C_{\max} \geq p_1 + (n-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + p_2 + t_2 + p_3,$$

and if scenario (b) is selected, then $C_{\max} \geq np_1 + \sum_{i=1}^{2} t_i + te_2 + t_2 + p_3 = LB_1'''(b1)$. Thus, the makespan value will not be less then

$$\min\left\{\sum_{i=1}^{3} p_i + (n-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + \sum_{i=1}^{2} t_i, np_1 + \sum_{i=1}^{2} t_i + te_2 + t_2 + p_3\right\} =$$

$$np_1 + \sum_{i=1}^{2} t_i + p_2 + p_3 + \min\left\{(t_2 + te_2) - p_2, (n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2) - p_1(n-1)\right\},$$

and

$$LB_1' = LB_1 + \min\left\{(t_2 + te_2) - p_2, (n-1)\left(p_2 - p_1 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)\right\}. \qquad (79)$$

Below, we provide an optimal schedule which matches the lower bound value in (79) for each of the following two different scenarios that can arise. The first (1.1.2(a)) is where $(t_2 + te_2) - p_2 \leq (n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_1)$ and the second (1.1.2 (b)) is the opposite.

For scenario 1.1.2(a), where $(t_2 + te_2) - p_2 \leq (n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_1)$, we define the following schedule (Schedule 1.1.2(a)):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S^m_{1j}, C^m_{1j}] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_j on M_2 during time interval $[S^m_{2j}, C^m_{2j}] = [jp_1 + t_1, S^m_{2j} + p_2]$ for $j = 1, ..., n$.
- Schedule job J_j on M_3 during time interval $[S^m_{3j}, C^m_{3j}] = [(j+1)p_1 + t_1 + t_2, S^m_{3j} + p_3]$ for $j = 1, ..., n-1$.
- Schedule job J_n on M_3 during time interval $[S^m_{3n}, C^m_{3n}] = [np_1 + t_1 + te_2 + 2t_2, S^m_{3n} + p_3]$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $[S^r_{1j}, C^r_{1j}] = [jp_1, jp_1 + t_1]$ for $j = 1, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $[S^r_{2j}, C^r_{2j}] = [(j+1)p_1 + t_1, S^r_{2j} + t_2]$ for $j = 1, ..., n-1$.
- Move job J_n from M_2 to M_3 during time interval $[S^r_{2n}, C^r_{2n}] = [np_1 + t_1 + t_2 + te_2, S^r_{2n} + t_2]$.
- Move the robot empty from M_2 to M_1 during time interval $[p_1 + t_1, p_1 + t_1 + te_1]$.

- Move the robot empty from M_3 to M_1 during time interval $[(j+1)p_1+t_1+t_2,(j+1)p_1+t_1+t_2+\sum_{i=1}^{2} te_i]$ for $j=1,...,n-2$.
- Move the robot empty from M_3 to M_2 during time interval $[np_1+t_1+t_2, np_1+t_1+t_2+te_2]$

Schedule 1.1.2(a) is illustrated in Figure 9 below for $n=4$ jobs.

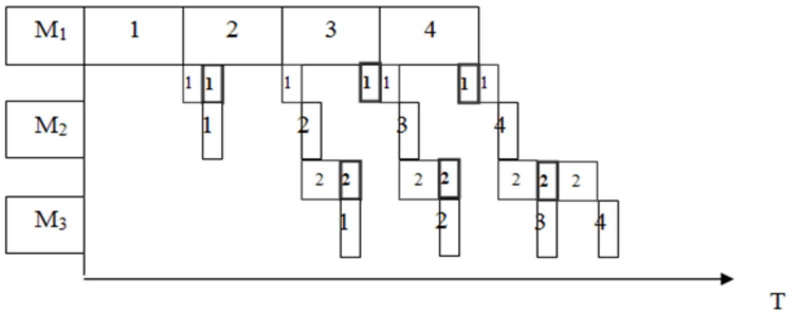

Figure 9: Schedule 1.1.2(a).

Lemma 9 *Schedule 1.1.2(a) is an optimal schedule for Scenario 1.1.2(a).*

Proof. We start by showing that schedule 1.1.2(a) is a feasible schedule. The fact that $S^m_{1,j+1}=C^m_{1j}=jp_1$; that $S^m_{2,j+1}=(j+1)p_1+t_1\geq C^m_{2j}=jp_1+t_1+p_2$; that $S^m_{3j}=(j+1)p_1+t_1+t_2\geq C^m_{3,j-1}=jp_1+t_1+t_2+p_3$ for $j=1,...,n-1$, and that $S^m_{3n}=np_1+t_1+te_2+2t_2\geq C^m_{3,n-1}=np_1+t_1+t_2+p_3$ follows from Case 1 condition that $p_1\geq\max\{p_2,p_3\}$ and from the fact that $te_2+t_2\geq p_3$. Thus, there is no overlap between two consecutive processing operations on the same machine. Moreover, the fact that $C^m_{1j}=S^r_{1j}=jp_1$; that $C^r_{1j}=S^m_{2j}=jp_1+t_1$; that $C^m_{2j}=jp_1+t_1+p_2\leq S^r_{2j}=(j+1)p_1+t_1$ for $j=1,...,n-1$; that $C^m_{2n}=np_1+t_1+p_2<S^r_{2n}=np_1+t_1+t_2+te_2$; that $C^r_{2j}=S^m_{3j}=(j+1)p_1+t_1+t_2$ for $j=1,...,n-1$; and that $C^r_{2n}=S^m_{3n}=np_1+t_1+2t_2+te_2$, where the first inequality follows from the fact that $p_1\geq p_2$ and the second inequality follows from the fact that $t_2+te_2>p_2$, implies that there is no overlap transferring operations of job J_j for $j=1,\ldots,n$. Thus, Condition 1 holds.

Once job J_1 is completed on machine M_1, the robot moves this job to M_2 during time interval $[p_1,p_1+t_1]$. Then, the robot moves empty from M_2 to M_1 during time interval $[p_1+t_1,p_1+t_1+te_1]$. Then, the robot performs the following set of moves one after the other. First, the robot moves job J_{j+1} from M_1 to M_2 during time interval $[(j+1)p_1,(j+1)p_1+t_1]$, followed by the move jobs J_j from M_2 to M_3 for $j = 1, \ldots, n-1$ is done during times interval $[(j+1)p_1 + t_1, (j+1)p_1 + \sum_{i=1}^{2} t_i]$, which then follows by a robot empty move from M_3 to M_1 during times interval $[(j+1)p_1 + \sum_{i=1}^{2} t_i, (j+1)p_1 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i]$ for $j = 1, \ldots, n-2$. Finally, the robot moves J_n from M_1 to M_2 during time interval $[np_1, np_1 + t_1]$, followed by move J_{n-1} from M_2 to M_3 during time interval $[np_1 + t_1, np_1 + t_1 + t_2]$. This operations are followed by an empty move from M_3 to M_2 during time interval $[np_1 + t_1 + t_2, np_1 + t_1 + t_2 + te_2]$, which is followed by the last move of job J_n from M_2 to M_3 during time interval $\left[np_1 + \sum_{i=1}^{2} t_i + te_2, np_1 + \sum_{i=1}^{2} t_i + te_2 + t_2\right]$. Therefore, there is no overlap between robot operations and they are all feasible. Thus, Condition 2 holds.

The fact that $C_{1j}^m = S_{1j}^r = jp_1$ for $j = 1, ..., n$ implies that job J_j is ready to move from M_1 to M_2 at time S_{1j}^r. Moreover, the fact that $C_{2j}^m = jp_1 + t_1 + p_2 \leq S_{2j}^r = (j+1)p_1 + t_1$ for $j = 1, ..., n-1$, and that $C_{2n}^m = np_1 + t_1 + p_2 \leq S_{2n}^r = np_1 + t_1 + t_2 + te_2$, where the first inequality follows from the fact that $p_1 \geq p_2$ and the second inequality follows from the fact that $t_2 + te_2 > p_2$, implies that job J_j is ready to move from M_2 to M_3 at time S_{2j}^r. Thus, Condition 3 holds.

It is implied from the feasibility of schedule 1.1.2(a) that the completion time of job n on M_3 is at time $C_{3n}^m = np_1 + \sum_{i=1}^{2} t_i + t_2 + te_2 + p_3$. The fact that this time matches the lower bound in (eq. 79) when $(t_2 + te_2) - p_2 \leq (n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_1)$ implies that this schedule is optimal for Scenario 1.1.2(a). ■

For scenario 1.1.2(b),where $(t_2 + te_2) - p_2 > (n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_1)$, we define the following schedule (Schedule 1.1.2(b)):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S_{1j}^m, C_{1j}^m] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_j on M_2 during time interval $[S_{2j}^m, C_{2j}^m] =$ $[p_1 + t_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), S_{2j}^m + p_2]$ for $j = 1, ..., n$.
- Schedule job J_j on M_3 during time interval $[S_{3j}^m, C_{3j}^m] =$ $[p_1 + t_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + p_2 + t_2, S_{3j}^m + p_3]$ for $j = 1, ..., n$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $[S_{1j}^r, C_{1j}^r] =$ $[p_1 + (j-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2\right), S_{1j}^r + t_1]$ for $j = 1, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $[S_{2j}^r, C_{2j}^r] =$ $[p_1 + t_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + p_2, S_{2j}^r + t_2]$ for $j = 1, ..., n$.

- Move the robot empty from M_3 to M_1 during time interval $[p_1+t_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2+t_2, p_1+t_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2+t_2+\sum_{i=1}^{2} te_i]$ for $j=1,...,n-1$.

Schedule 1.1.2(b) is illustrated in Figure 10 below for $n=3$ jobs with empty return moves shown in bold.

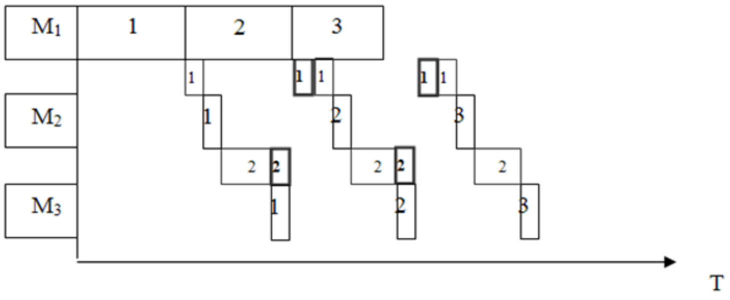

Figure 10: Schedule 1.1.2(b).

Lemma 10 *Schedule 1.1.2(b) is an optimal schedule for Scenario 1.1.2(b).*

Proof. We start by showing that schedule 1.1.2(b) is a feasible schedule. The fact that $S_{1,j+1}^{m}=C_{1j}^{m}=jp_1$; that

$$S_{2,j+1}^{m}=p_1+t_1+j\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)>C_{2j}^{m}=p_1+t_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2;$$

and that

$$\begin{aligned} S_{3,j+1}^{m} &= p_1+t_1+j\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2+t_2> \\ C_{3j}^{m} &= p_1+t_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2+t_2+p_3, \end{aligned}$$

where the last inequality follows from the fact that $p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i>p_1\geq p_3$, implies that there are no overlap between processing of jobs in any machine. Moreover, the fact that $C_{1j}^{m}=jp_1\leq S_{1j}^{r}=p_1+(j-1)\left(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i+p_2\right)$; that $C_{1j}^{r}=S_{2j}^{m}=p_1+(j-1)\left(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i+p_2\right)+t_1$; that $C_{2j}^{m}=S_{2j}^{r}=p_1+t_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2$; and that $C_{2j}^{r}=S_{3j}^{m}=p_1+t_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2+t_2$, implies that there is no overlap between transferring operations of job J_j for $j = 1, \dots n$. Thus, Condition 1 holds.

Once job J_j $(j = 1, ..., n)$ is completed on machine M_1, the robot perform move J_j from M_1 to M_2 during time interval

$$[p_1 + (j-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2\right), p_1 + (j-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2\right) + t_1]$$

followed by a move J_j from M_2 to M_3 during time interval

$$[p_1+t_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2, p_1+t_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2+t_2]$$

and for $j = 1, ..., n-1$ this move is followed by an empty move of the robot from M_3 to M_1 during time interval

$$[p_1 + t_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + p_2 + t_2,$$
$$p_1 + t_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + p_2 + t_2 + \sum_{i=1}^{2} te_i].$$

Therefore, there is no overlap between robot operations and Condition 2 holds.

The fact that $C_{1j}^m = jp_1 \leq S_{1j}^r = p_1 + (j-1)\left(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2\right)$ implies that job J_j is ready to move from M_1 to M_2 at time S_{1j}^r. Moreover, the fact that $C_{2j}^m = S_{2j}^r = p_1 + t_1 + (j-1)\left(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) + p_2$ implies that job J_j is ready to move from M_2 to M_3 at time S_{2j}^r. Thus, Condition 3 holds.

It is implied from the feasibility of schedule 1.1.2(b) that the completion time of job n on machine M_3 is at time

$$C_{3n}^m = p_1 + t_1 + (n-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + p_2 + t_2 + p_3.$$

The fact that this time matches the lower bound in (eq. 79) when $(n-1)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2 - p_1) < (t_2 + te_2) - p_2$ implies that this schedule 1.1.2(b) is optimal for Scenario 1.1.2(b).
■

The analysis of subcase 1.1.3 Analyses of subcase 1.1.3, where $p_1 < p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i$ and $p_3 = \max\{p_2, p_3, t_2 + te_2\}$ is presented in subsection 4.4.1.

9.1.2 The analysis of subcase 1.2

For subcase 1.2, where $p_1 \geq p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$, we define the following schedule (Schedule 1.2):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S_{1j}^m, C_{1j}^m] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_j on M_2 during time interval $[S_{2j}^m, C_{2j}^m] = [jp_1 + t_1, jp_1 + t_1 + p_2]$ for $j = 1, ..., n$.
- Schedule job J_j on M_3 during time interval $[S_{3j}^m, C_{3j}^m] = [jp_1 + p_2 + \sum_{i=1}^{2} t_i, jp_1 + p_2 + p_3 + \sum_{i=1}^{2} t_i]$ for $j = 1, ..., n$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $[S_{1j}^r, C_{1j}^r] = [jp_1, jp_1 + t_1]$ for $j = 1, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $[S_{2j}^r, C_{2j}^r] = [jp_1 + t_1 + p_2, jp_1 + p_2 + \sum_{i=1}^{2} t_i]$ for $j = 1, ..., n$.
- Move the robot empty from M_3 to M_1 during time interval $[jp_1 + p_2 + \sum_{i=1}^{2} t_i, jp_1 + p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i]$ for $j = 1, ..., n-1$.

Schedule 1.2 is illustrated in Figure 11 below for $n = 4$ jobs with empty return moves shown in bold.

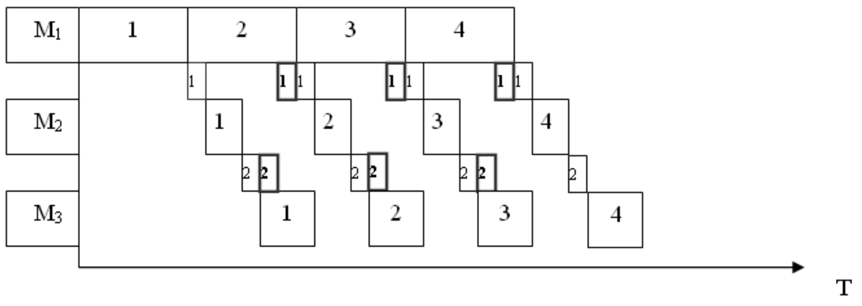

Figure 11: Schedule 1.2.

Lemma 11 *Schedule 1.2 is an optimal schedule for subcase 1.2.*

Proof. We start by showing that Schedule 1.2 is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$; that $S^m_{2,j+1} = (j+1)p_1 + t_1 > C^m_{2j} = jp_1 + t_1 + p_2$; and that

$$S^m_{3,j+1} = (j+1)p_1 + p_2 + \sum_{i=1}^{2} t_i > C^m_{3j} = jp_1 + \sum_{i=1}^{2} t_i + p_2 + p_3,$$

where the two last inequalities follow from the fact that for *Case 1* we have that both $p_1 > p_2$ and $p_1 > p_3$, implying that there is no overlap between processing operations on the same machine. Moreover, the fact that $C^m_{1j} = S^r_{1j} = jp_1$; that $C^r_{1j} = S^m_{2j} = jp_1 + t_1$; that $C^m_{2j} = S^r_{2j} = jp_1 + t_1 + p_2$; and that $C^r_{2j} = S^m_{3j} = jp_1 + p_2 + \sum_{i=1}^{2} t_i$, implies that there is no overlap between processing and transferring operations of job J_j for $j = 1, \ldots n$. Thus, Condition 1 holds.

Once job J_j is completed on machine M_1, the robot moves this job to M_2 during time interval $[jp_1, jp_1 + t_1]$ and then waits beside M_1 for p_2 units of time. After that the robot transfers job J_j from machine M_2 to machine M_3 during time interval $[jp_1 + t_1 + p_2, jp_1 + p_2 + \sum_{i=1}^{2} t_i]$. At the end of this operation the robot moves empty from M_3 to M_1, so it arrives to machine M_1 at time point $jp_1 + p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$. The fact that $jp_1 + p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i \leq S^r_{1,j+1} = (j+1)p_1$ follows from the case condition that $p_1 \geq p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$, and implies that there is no overlap between robot operations. Thus, Condition 2 holds.

The fact that $C^m_{1j} = S^r_{1j} = jp_1$ implies that job J_j is ready to move from M_1 to M_2 at time S^r_{1j}. Moreover, the fact that $C^m_{2j} = S^r_{2j} = jp_1 + t_1 + p_2$ implies that job J_j is ready to move from M_2 to M_3 at time S^r_{2j}. Thus, Condition 3 holds.

It is implied from the feasibility of Schedule 1.1 that the completion time of job n on machine M_3 is at time $C^m_{3n} = np_1 + p_2 + p_3 + \sum_{i=1}^{2} t_i$. The fact that this time matches the lower bound value in eq. (9) implies that this Schedule 1.2 is optimal for subcase 1.2. ∎

9.2 Optimal schedules for *Case 2*

For this case, where $p_2 = \max\{p_1, p_2, p_3, \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\}$ (i.e., condition in (6) for $f = 2$ holds), we suggest the following schedule (Schedule 2):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S_{1j}^m, C_{1j}^m] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_j on M_2 during time interval $[S_{2j}^m, C_{2j}^m] = [p_1 + t_1 + (j-1)p_2, p_1 + t_1 + jp_2]$ for $j = 1, ..., n$.
- Schedule job J_j on M_3 during time interval $[S_{3j}^m, C_{3j}^m] = [p_1 + t_1 + jp_2 + t_2, p_1 + t_1 + jp_2 + t_2 + p_3]$ for $j = 1, ..., n$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $[S_{1j}^r, C_{1j}^r] = [p_1 + (j-1)p_2, p_1 + (j-1)p_2 + t_1]$ for $j = 1, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $[S_{2j}^r, C_{2j}^r] = [p_1 + jp_2 + t_1, p_1 + jp_2 + t_1 + t_2]$ for $j = 1, ..., n$.
- Move the robot empty from M_2 to M_1 during time interval $[p_1 + t_1, p_1 + t_1 + te_1]$.
- Move the robot empty from M_3 to M_1 during time interval $[p_1 + t_1 + t_2 + (j-1)p_2, p_1 + t_1 + t_2 + (j-1)p_2 + te_2 + te_1]$ for $j = 2, ..., n-1$.
- Move the robot empty from M_3 to M_2 during time interval $[p_1 + t_1 + (n-1)p_2 + t_2, p_1 + t_1 + (n-1)p_2 + t_2 + te_2]$.

Schedule 2 is illustrated in Figure 12 below for $n = 4$ with empty return moves shown in bold.

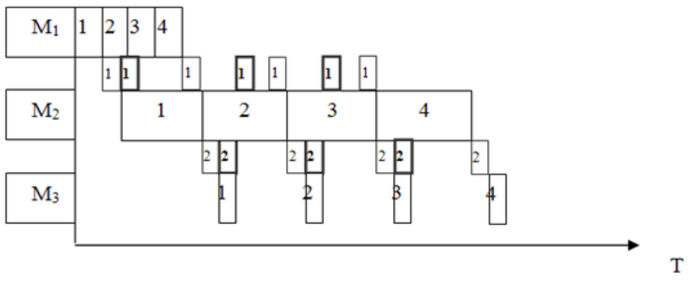

Figure 12: Schedule 2.

Lemma 12 *Schedule 2 is an optimal schedule for* Case 2.

Proof. We start by showing that Schedule 2 is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$; that $S^m_{2,j+1} = C^m_{2j} = p_1 + t_1 + jp_2$; and that $S^m_{3,j+1} = p_1 + (j+1)p_2 + t_1 + t_2 \geq C^m_{3j} = p_1 + jp_2 + t_1 + t_2 + p_3$, where the last inequality follows from the fact that $p_2 \geq p_3$, implies that there is no overlap between the processing of different jobs on each machine. Moreover, the fact that $C^m_{1j} = jp_1 \leq S^r_{1j} = p_1 + (j-1)p_2$; that $C^r_{1j} = S^m_{2j} = p_1 + (j-1)p_2 + t_1$; that $C^m_{2j} = S^r_{2j} = p_1 + jp_2 + t_1 + t_2$; and that $C^r_{2j} = S^m_{3j} = p_1 + t_1 + jp_2 + t_2$, where the first inequality follows from the fact that $p_2 \geq p_1$, implies that there is no overlap between processing and transferring operations of job J_j for $j = 1, ..., n$. Thus, Condition 1 holds.

Once job J_1 is completed on M_1, the robot moves this job to M_2 during time interval $[p_1, p_1 + t_1]$ and returns empty to M_1 during time interval $[p_1 + t_1, p_1 + t_1 + te_1]$. Then, for $j = 2, ..., n$, the robot moves job J_j from M_1 to M_2 during time period $[p_1 + (j-1)p_2, p_1 + (j-1)p_2 + t_1]$; job J_{j-1} from M_2 to M_3 during time interval $[p_1 + (j-1)p_2 + t_1, p_1 + (j-1)p_2 + t_1 + t_2]$ and returns empty to M_1 during time interval $[p_1 + t_1 + t_2 + (j-1)p_2, p_1 + t_1 + t_2 + (j-1)p_2 + te_2 + te_1]$. The fact that $p_1 + t_1 + te_1 \leq p_1 + p_2$, and that $C^r_{1j} = S^r_{2,j-1} = p_1 + (j-1)p_2 + t_1$, implies that there is no overlap between those operations. Moreover, the fact that $p_1 + (j-1)p_2 + t_1 + t_2 + te_2 + te_1 \leq p_1 + jp_2$ implies that the robot will be ready to move job J_{j+1} from M_1 to M_2 at time $p_1 + jp_2$. Lastly, after downloading job J_{n-1} at M_3 at time $p_1 + (n-1)p_2 + t_1 + t_2$, it returns empty to M_2 during time interval $[p_1 + (n-1)p_2 + t_1 + t_2, p_1 + (n-1)p_2 + t_1 + t_2 + te_2]$ and transfers job J_n from M_2 to M_3 during time interval $[p_1 + np_2 + t_1, p_1 + np_2 + t_1 + t_2]$. The fact that $p_1 + (n-1)p_2 + t_1 + t_2 + te_2 \leq p_1 + np_2 + t_1$ completes our proof that there is no overlap between robot operations and they are all feasible. Thus, Condition 2 holds.

The fact that $C^m_{1j} = jp_1 \leq S^r_{1j} = p_1 + (j-1)p_2$ follows from the fact that $p_2 \geq p_1$, and implies that job J_j is ready to move from M_1 to M_2 at time $S^r_{1j} = p_1 + (j-1)p_2$. Moreover, the fact that $C^m_{2j} = p_1 + jp_2 + t_1 = S^r_{2j}$ implies that job J_j is ready to move from M_2 to M_3 at time S^r_{2j} and thus Condition 3 holds as well and Schedule 2 is feasible.

It is implied from the feasibility of Schedule 2 that the completion time of job n on M_3 is at time $C^m_{3n} = p_1 + np_2 + t_1 + t_2 + p_3$. The fact that this time matches the lower bound in (3) for $f = 2$ implies that this Schedule 2 is optimal for Case 2. ■

9.3 The analysis of *Case 3*

We further divide this case, where $p_3 = \max\{p_1, p_2, p_3, \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\}$, to two subcases. Subcase 3.1 is where $p_3 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$, and subcase 3.2 is the opposite case. The analysis of these two subcases appears below. Note that Case 3 presents more subcases than presented in section 4.4, in order to provide simpler proofs of the optimally of the schedule.

9.3.1 The analysis of subcase 3.1

We further divide this subcase, where $p_3 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$, to three subcases. The first subcase 3.1.1 is where $p_2 = \max\{p_1, t_1 + te_1, p_2\}$, the second subcase 3.1.2 is where $p_1 = \max\{p_1, t_1 + te_1, p_2\}$, and the last subcase is where $t_1 + te_1 = \max\{p_1, t_1 + te_1, p_2\}$.

The analysis of subcase 3.1.1 For this subcase, where $p_3 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ and $p_2 =$
$\max\{p_1, t_1 + te_1, p_2\}$, we further partition this subcase to two subcases. The first subcase 3.1.1.1 is where $p_2 \geq \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ and the second subcase 3.1.1.2 is the opposite case.

Optimal schedule for subcase 3.1.1.1 For subcase 3.1.1.1, where $p_3 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ and $p_2 = \max\{p_1, t_1 + te_1, p_2\}$ and $p_2 \geq \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$, we define the following schedule (Schedule 3.1.1.1):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S_{1j}^m, C_{1j}^m] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_j on M_2 during time interval $[S_{2j}^m, C_{2j}^m] = [p_1 + t_1 + (j-1)p_2, p_1 + t_1 + jp_2]$ for $j = 1, 2, ..., n$.
- Schedule job J_j on M_3 during time interval $[S_{3j}^m, C_{3j}^m] = [p_1 + t_1 + p_2 + t_2 + (j-1)p_3, p_1 + t_1 + p_2 + t_2 + jp_3]$ for $j = 1, ..., n$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $[S_{1j}^r, C_{1j}^r] = [p_1 + (j-1)p_2, p_1 + (j-1)p_2 + t_1]$ for $j = 1, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $[S_{2j}^r, C_{2j}^r] = [p_1 + jp_2 + t_1, p_1 + jp_2 + t_1 + t_2]$ for $j = 1, ..., n$.
- Move the robot empty from M_2 to M_1 during time interval $[p_1 + t_1, p_1 + t_1 + te_1]$.
- Move the robot empty from M_3 to M_1 during time interval $[p_1 + (j-1)p_2 + t_1 + t_2, p_1 + (j-1)p_2 + t_1 + t_2 + \sum_{i=1}^{2} te_i]$ for $j = 2, ..., n-1$.

- Move the robot empty from M_3 to M_2 during time interval $[p_1+(n-1)p_2+t_1+t_2, p_1+(n-1)p_2+t_1+t_2+te_2]$.

Schedule 3.1.1.1 is illustrated in Figure 13 below for $n = 3$ with empty return moves shown in bold.

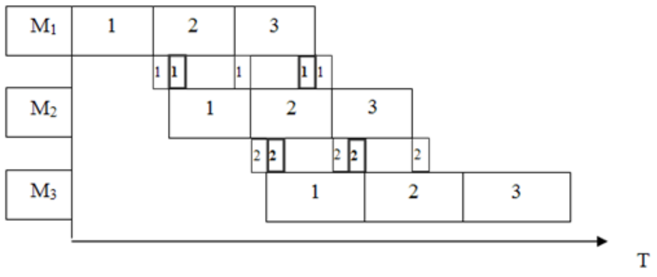

Figure 13: Schedule 3.1.1.1.

Lemma 13 *Schedule 3.1.1.1 is an optimal schedule for subcase 3.1.1.1.*

Proof. We start by showing that Schedule 3.1.1.1 is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$; that $S^m_{2,j+1} = C^m_{2j} = p_1 + t_1 + jp_2$; and that $S^m_{3,j+1} = C^m_{3j} = p_1 + t_1 + p_2 + t_2 + jp_3$, implies that there is no overlap between the processing of different jobs on each of the machines. Moreover, the fact that $C^m_{1j} = jp_1 \leq S^r_{1j} = p_1 + (j-1)p_2$; that $C^r_{1j} = S^m_{2j} = p_1 + (j-1)p_2 + t_1$; that $C^m_{2j} = S^r_{2j} = p_1 + jp_2 + t_1$; and that $C^r_{2j} = p_1+jp_2+t_1+t_2 \leq S^m_{3j} = p_1+t_1+p_2+t_2+(j-1)p_3$, implies that there is no overlap between processing and transferring operations of job J_j for $j = 1, ..., n$. Thus, Condition 1 holds.

Once job J_1 is completed on M_1, the robot moves this job to M_2 during time interval $[p_1, p_1+t_1]$ and returns empty to M_1 during time interval $[p_1+t_1, p_1+t_1+te_1]$. Then, for $j = 2, ..., n$, the robot moves job J_j from M_1 to M_2 during time period $[p_1+(j-1)p_2, p_1+(j-1)p_2+t_1]$; job J_{j-1} from M_2 to M_3 during time interval $[p_1+(j-1)p_2+t_1, p_1+(j-1)p_2+t_1+t_2]$, and returns empty to M_1 during time interval $[p_1+t_1+t_2+(j-1)p_2, p_1+t_1+t_2+(j-1)p_2+te_2+te_1]$. The fact that $p_1 + t_1 + te_1 \leq p_1 + p_2$, and that $C^r_{1j} = S^r_{2j-1} = p_1 + (j-1)p_2 + t_1$ implies that there is no overlap between those operations. Moreover, the fact that $p_1 + (j-1)p_2 + t_1 + t_2+te_2+te_1 \leq p_1+jp_2$, implies that the robot will be ready to move job J_{j+1} from M_1 to M_2 at time p_1+jp_2. Lastly, after downloading job J_{n-1} at M_3 at time $p_1+(n-1)p_2+t_1+t_2$, it returns empty to M_2 during time interval $[p_1+(n-1)p_2+t_1+t_2, p_1+(n-1)p_2+t_1+t_2+te_2]$ and transfers job J_n from M_2 to M_3 during time interval $[p_1 + np_2 + t_1, p_1 + np_2 + t_1 + t_2]$.

The fact that $p_1 + (n-1)p_2 + t_1 + t_2 + te_2 \leq p_1 + np_2 + t_1$ completes our proof that there is no overlap between robot operations and they are all feasible. Thus, Condition 2 holds.

The fact that $C^m_{1j} = jp_1 \leq S^r_{1j} = p_1 + (j-1)p_2$ follows from the fact that $p_2 \geq p_1$ and implies that job J_j is ready to move from M_1 to M_2 at time S^r_{1j}. Moreover, the fact that $C^m_{2j} = p_1 + jp_2 + t_1 = S^r_{2j}$ implies that job J_j is ready to move from M_2 to M_3 at time S^r_{2j}. Thus, Condition 3 holds as well and Schedule 3.1.1.1 is feasible.

It is implied from the feasibility of schedule 3.1.1.1 that the completion time of job n on M_3 is at time $C^m_{3n} = p_1 + t_1 + p_2 + t_2 + np_3$. The fact that this time matches the lower bound in (20) implies that this schedule 3.1.1.1 is optimal for subcase 3.1.1.1. ∎

Optimal schedule for subcase 3.1.1.2 For subcase 3.1.1.2, where $p_3 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ and $p_2 = \max\{p_1, t_1 + te_1, p_2\}$ and $p_2 < \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$, we define the following schedule (Schedule 3.1.1.2):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S^m_{1j}, C^m_{1j}] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_1 on M_2 during time interval $[S^m_{21}, C^m_{21}] = [p_1 + t_1, p_1 + t_1 + p_2]$.
- Schedule job J_j on M_2 during time interval $[S^m_{2j}, C^m_{2j}] = [p_1 + t_1 + p_2 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), p_1 + t_1 + 2p_2 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)]$ for $j = 2, ..., n$.
- Schedule job J_j on M_3 during time interval $[S^m_{3j}, C^m_{3j}] = [p_1 + t_1 + p_2 + t_2 + (j-1)p_3, p_1 + t_1 + p_2 + t_2 + jp_3]$ for $j = 1, ..., n$.

Robot Schedule:

- Move job J_1 from M_1 to M_2 during time interval $[S^r_{11}, C^r_{11}] = [p_1, p_1 + t_1]$.
- Move job J_j from M_1 to M_2 during time interval $[S^r_{1j}, C^r_{1j}] = [p_1 + p_2 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), p_1 + p_2 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1]$ for $j = 2, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $[S^r_{2j}, C^r_{2j}] = [p_1 + t_1 + p_2 + (j-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), p_1 + t_1 + p_2 + (j-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_2]$ for $j = 1, ..., n$.
- Move the robot empty from M_2 to M_1 during time interval $[p_1 + t_1, p_1 + t_1 + te_1]$.
- Move the robot empty from M_3 to M_1 during time interval $[p_1 + p_2 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + t_2, p_1 + p_2 + (j-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)]$ for $j = 2, ..., n-1$.

- Move the robot empty from M_3 to M_2 during time interval $[p_1 + p_2 + (n-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + t_2, p_1 + p_2 + (n-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + t_2 + te_2]$.

Schedule 3.1.1.2 is illustrated in Figure 14 below for $n = 3$ with empty return moves shown in bold.

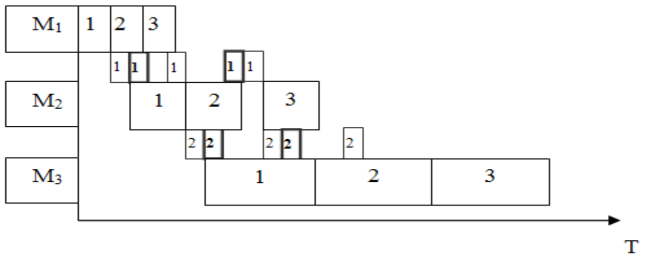

Figure 14: Schedule 3.1.1.2.

Lemma 14 *Schedule 3.1.1.2 is an optimal schedule for subcase 3.1.1.2.*

Proof. We start by showing that Schedule 3.1.1.2 is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$; that $S^m_{22} = C^m_{21} = p_1 + t_1 + p_2$; that

$$S^m_{2,j+1} = p_1 + t_1 + p_2 + (j-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) > C^m_{2j} = p_1 + t_1 + 2p_2 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$$

for $j = 2, ..., n-1$; and that $S^m_{3,j+1} = C^m_{3j} = p_1 + t_1 + p_2 + t_2 + jp_3$, implies that there is no overlap between the processing of different jobs on each of the machines (the inequality follows from the fact that $p_2 < \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$). Moreover, the fact that $C^m_{11} = S^r_{11} = p_1$; that $C^m_{1j} = jp_1 < S^r_{1j} = p_1 + p_2 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$ (the inequality follows from the fact $p_1 \leq p_2 < \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$); that $C^r_{1j} = S^m_{2j} = p_1 + p_2 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1$; that $C^m_{21} = S^r_{21} = p_1 + t_1 + p_2$; that

$$C^m_{2j} = p_1 + t_1 + 2p_2 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) < S^r_{2j} = p_1 + t_1 + p_2 + (j-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$$

(the inequality follows from the fact $p_2 < \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$); and that

$$C^r_{2j} = p_1 + t_1 + p_2 + (j-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_2 \leq S^m_{3j} = p_1 + t_1 + p_2 + t_2 + (j-1)p_3$$

(the inequality follows from the fact $p_3 \geq \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$) implies that there is no overlap between processing and transferring operations of job J_j for $j = 1, ..., n$. Thus, Condition 1 holds.

Once job J_1 is completed on M_1, the robot moves this job to M_2 during time interval $[p_1, p_1 + t_1]$ and returns empty to M_1 during time interval $[p_1 + t_1, p_1 + t_1 + te_1]$. Then, for $j = 2, ..., n$, the robot moves job J_j from M_1 to M_2 during time period

$$[p_1 + p_2 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), p_1 + p_2 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1];$$

job J_{j-1} from M_2 to M_3 during time interval

$$[p_1 + t_1 + p_2 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), p_1 + t_1 + p_2 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_2],$$

and returns empty to M_1 during time interval

$$[p_1+p_2+(j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)+t_1+t_2, p_1+p_2+(j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)+t_1+t_2+\sum_{i=1}^{2} te_i].$$

The fact that $p_1+t_1+te_1 \leq p_1+p_2$, and that $C_{1j}^r = S_{2,j-1}^r = p_1+p_2+(j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$ $+t_1$ implies that there is no overlap between those operations. Moreover, the fact that the robot returns to machine M_1 at time $p_1 + p_2 + (j-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$ implies that the robot will be ready to move job J_j from M_1 to M_2 at that time. Lastly, after downloading job J_{n-1} at M_3 at time $p_1 + t_1 + p_2 + (n-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_2$, it returns empty to M_2 during time interval

$$[p_1+p_2+(n-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)+t_1+t_2, p_1+p_2+(n-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)+t_1+t_2+te_2]$$

and transfer job J_n from M_2 to M_3 during time interval

$$[p_1 + t_1 + p_2 + (n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), p_1 + t_1 + p_2 + (n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_2].$$

The fact that

$$p_1 + p_2 + (n-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + t_2 + te_2 < p_1 + t_1 + p_2 + (n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$$

completes our proof that there is no overlap between robot operations and they are all feasible. Thus, Condition 2 holds.

The fact that $C^m_{11} = S^r_{11} = p_1$ and that $C^m_{1j} = jp_1 \leq S^r_{1j} =$ $p_1 + p_2 + (j-2)\left(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right)$ follows from the fact that $p_1 \leq p_2 < \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i$ and implies that job J_j is ready to move from M_1 to M_2 at time S^r_{1j} for $j = 1, ..., n$. Moreover, the fact that

$$C^m_{2j} = p_1 + t_1 + 2p_2 + (j-2)\left(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) < S^r_{2j} = p_1 + t_1 + p_2 + (j-1)\left(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right)$$

follows from the fact that $p_2 < \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i$ and implies that job J_j is ready to from M_2 to M_3 at time S^r_{2j} and thus Condition 3 holds as well and Schedule 3.1.1.2 is feasible.

It is implied from the feasibility of schedule 3.1.1.2 that the completion time of job n on M_3 is at time $C^m_{3n} = p_1 + t_1 + p_2 + t_2 + np_3$. The fact that this time matches the lower bound in (20) implies that this schedule 3.1.1.2 is optimal for subcase 3.1.1.2. ■

The analysis of subcase 3.1.2 In subcase 3.1.2, where $p_3 < p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i$ and $p_1 = \max\{p_1, t_1 + te_1, p_2\}$, we first update LB_3 and provide a revised (tighter) bound denoted by LB_3''''. We consider two possible scenarios: (a) the robot does not perform two consecutive moves from M_1 to M_2, and (b) at least once the robot makes two consecutive moves from M_1 to M_2. Let us first consider case (a). In this case the robot moves each job from M_1 to M_2 and then to M_3 before moving to the next job, i.e., after transferring job J_j $(j = 1, ..., n)$ from M_1 to M_2, the robot waits beside M_2 for the completion of J_j on this machine and then moves it to M_3. This thus implies that the robot will start to move J_1 from M_1 to M_2 not earlier than p_1, and will start to move J_j from M_1 to M_2 not earlier than $p_1 + (j-1)\left(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right)$ for $j = 2, ..., n$. Thus, job J_n will finish its processing on M_3 not earlier than at time

$$LB_3''''(a) = p_1 + (n-1)\left(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) + t_1 + p_2 + t_2 + p_3 =$$
$$LB_3 + (n-1)\left(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i - p_3\right) \geq LB_3.$$

Consider the next case (b), where at least once the robot makes two consecutive moves from M_1 to M_2. We further divide case (b) to two subcases. The first $(b1)$, where the *first* two consecutive moves from M_1 to M_2 are done on jobs J_1 and J_2, and the second $(b2)$, where the *first* two consecutive moves from M_1 to M_2 are done on jobs J_x and J_{x+1} with $2 \leq x \leq n-1$.

In subcase $(b1)$, the robot will start moving J_2 from M_1 to M_2 not earlier than at time $2p_1$, which further implies that the move of J_1 from M_2 to M_3 will not start before $2p_1+t_1$. Thus, job J_1 will not start its processing on M_3 before $2p_1+t_1+t_2$, and the makespan value will be not less than

$$LB_3''''(b1) = 2p_1 + t_1 + t_2 + np_3 = LB_3 + (p_1 - p_2) \geq LB_3.$$

In subcase $(b2)$, the move of J_{x-1} from M_2 to M_3 will not finish before time $p_1 + (x-2)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + p_2 + t_2$. Thus, the move of J_x from M_1 to M_2 will not start before $p_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$, which further implies that the move of J_{x+1} from M_1 to M_2 will not start before

$$\max\left\{p_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + te_1, (x+1)p_1\right\}.$$

This further implies that the move of J_x from M_2 to M_3 cannot start before

$$\max\left\{p_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + te_1, (x+1)p_1\right\} + t_1,$$

which is the earliest time that the move of J_{x+1} from M_1 to M_2 is completed and before $p_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + p_2$ which is the earliest time job J_x is completed on M_2. Thus, the move of J_x from M_2 to M_3 cannot start before

$$p_1 + \max\{\max\{(x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + te_1, xp_1\} + t_1,$$
$$(x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + p_2\}$$

$$= p_1 + \max\{(x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + \max\{t_1 + te_1, p_2\}, xp_1 + t_1\}$$

and the makespan value in this case is lower bounded by

$$LB_3''''(b2) = p_1 + \max\{(x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + \max\{t_1 + te_1, p_2\}, xp_1 + t_1\}$$
$$+ t_2 + (n-x+1)p_3 \geq LB_3.$$

Let x^* be the x value that minimizes $LB_3''''(b2)$ (note that x^* can be easily determined in $O(n)$ time) and let $LB_3''''(b2(x^*)) = LB_3 + \Delta(x^*)$. If scenario (a) is selected, then

$$C_{\max} \geq p_1 + (n-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + p_2 + t_2 + p_3,$$

and if scenario (b) is selected, then $C_{\max} \geq \min\{LB_3''''(b1), LB_3''''(b2)\}$. Thus, the makespan value will not be less than

$$LB_3'''' = LB_3 + \min\left\{(n-1)\left(\textstyle\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3\right), p_1 - p_2, \Delta(x^*)\right\}. \tag{80}$$

Below, we provide an optimal schedule that matches the lower bound value in (80) for each of the following three scenarios that can arise. The first subcase 3.1.2.1 is where $(n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3)$ is minimum, the second subcase 3.1.2.2 is where $p_1 - p_2$ is minimum, and third subcase 3.1.2.3 is where $\Delta(x^*)$ is minimum out, of the three terms in (80).

Optimal schedule for subcase 3.1.2.1 For subcase 3.1.2.1, where $p_3 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ and $p_1 = \max\{p_1, t_1 + te_1, p_2\}$ and

$$\min\left\{(n-1)\left(\textstyle\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3\right), p_1 - p_2, \Delta(x^*)\right\} = (n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3),$$

we define the following schedule (Schedule 3.1.2.1):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S_{1j}^m, C_{1j}^m] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_j on M_2 during time interval $[S_{2j}^m, C_{2j}^m] = [p_1 + t_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i), S_{2j}^m + p_2]$ for $j = 1, ..., n$.
- Schedule job J_j on M_3 during time interval $[S_{3j}^m, C_{3j}^m] = [\sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i), S_{3j}^m + p_3]$ for $j = 1, ..., n$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $[S^r_{1j}, C^r_{1j}] = [p_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i), S^r_{1j} + t_1]$ for $j = 1, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $[S^r_{2j}, C^r_{2j}] = [\sum_{i=1}^{2} p_i + t_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i), S^r_{2j} + t_2]$ for $j = 1, ..., n$.
- Move the robot empty from M_3 to M_1 during time interval $[\sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i), \sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + \sum_{i=1}^{2} te_i]$ for $j = 1, ..., n-1$.

Schedule 3.1.2.1 is illustrated in Figure 15 below for $n = 4$ with empty return moves shown in bold.

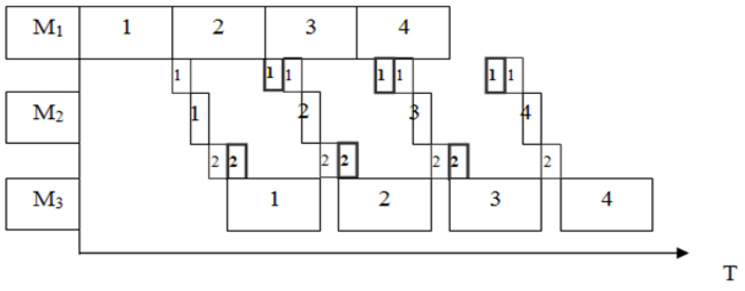

Figure 15: Schedule 3.1.2.1.

Lemma 15 *Schedule 3.1.2.1 is an optimal schedule for subcase 3.1.2.1.*

Proof. We start by showing that schedule 3.1.2.1 is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$; that

$$S^m_{2,j+1} = p_1 + t_1 + j(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) > C^m_{2j} = p_1 + t_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + p_2;$$

and that

$$\begin{aligned} S^m_{3,j+1} &= \sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + j(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) > C^m_{3j} = \\ &\sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + p_3, \end{aligned}$$

implies that there are no overlaps between processing of jobs in any machine. Moreover, the fact that $C^m_{1j} = jp_1 \le S^r_{1j} = p_1 + (j-1)(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i)$; that $C^r_{1j} = S^m_{2j} = p_1 + (j-1)(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i) + t_1$; that $C^m_{2j} = S^r_{2j} = \sum_{i=1}^2 p_i + t_1 + (j-1)(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i)$; and that $C^r_{2j} = S^m_{3j} = \sum_{i=1}^2 p_i + \sum_{i=1}^2 t_i + (j-1)(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i)$, implies that there is no overlap between processing and transferring operations of job J_j for $j = 1, \dots n$. Thus, Condition 1 holds.

Since

$$C^r_{1j} = p_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + t_1 < S^r_{2j} = \sum_{i=1}^{2} p_i + t_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i),$$

there is no overlap between moves J_j from M_1 to M_2 and J_j from M_2 to M_3. Moreover, after completing the move of J_j from M_2 to M_3, the robot returns to M_1 at time

$$\sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + \sum_{i=1}^{2} te_i = S^r_{1,j+1},$$

which implies that there is no overlap between moves of J_j from M_2 to M_3 and J_{j+1} from M_1 to M_2. Therefore, there is no overlap between robot operations and Condition 2 holds.

The fact that $C^m_{1j} = jp_1 \le S^r_{1j} = p_1 + (j-1)(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i)$ implies that job J_j is ready to move from M_1 to M_2 at time S^r_{1j}. Moreover, the fact that $C^m_{2j} = S^r_{2j} = \sum_{i=1}^2 p_i + t_1 + (j-1)(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i)$, implies that job J_j is ready to move from M_2 to M_3 at time S^r_{2j}. Thus, Condition 3 holds.

It is implied from the feasibility of schedule 3.1.2.1 that the completion time of job J_n on machine M_3 is at time

$$C^m_{3n} = \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} p_i + (n-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + p_3.$$

The fact that this time matches the lower bound in (eq. 80) when

$$(n-1)\left(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2 - p_3\right) \le \min\{p_1 - p_2, \Delta(x^*)\},$$

implies that this schedule 3.1.2.1 is optimal for subcase 3.1.2.1. ■

Optimal schedule for subcase 3.1.2.2 For subcase 3.1.2.2, where $p_3 < p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i$ and $p_1 = \max\{p_1, t_1 + te_1, p_2\}$ and

$$\min\left\{(n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3\right), p_1 - p_2, \Delta(x^*)\right\} = p_1 - p_2,$$

we define the following schedule (Schedule 3.1.2.2):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S_{1j}^m, C_{1j}^m] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_j on M_2 during time interval $[S_{2j}^m, C_{2j}^m] = [jp_1 + t_1, S_{2j}^m + p_2]$ for $j = 1, 2$.
- Schedule job J_j on M_2 during time interval $[S_{2j}^m, C_{2j}^m] =$ $\left[\max\left\{2p_1 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), jp_1\right\} + t_1, S_{2j}^m + p_2\right]$ for $j = 3, ..., n$.
- Schedule job J_j on M_3 during time interval $[S_{3j}^m, C_{3j}^m] = [2p_1 + t_1 + t_2 + (j-1)p_3, S_{3j}^m + p_3]$ for $j = 1, ..., n$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $[S_{1j}^r, C_{1j}^r] = [jp_1, jp_1 + t_1]$ for $j = 1, 2$.
- Move job J_j from M_1 to M_2 during time interval $[S_{1j}^r, C_{1j}^r] = [\max\{S_{2,j-1}^m + t_2 + \sum_{i=1}^{2} te_i, jp_1\}, S_{1j}^r + t_1]$ for $j = 3, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $[S_{2j}^r, C_{2j}^r] = [S_{2,j+1}^m, S_{2j}^r + t_2]$ for $j = 1, ..., n-1$.
- Move job J_n from M_2 to M_3 during time interval $[S_{2n}^r, C_{2n}^r] = [S_{2n}^m + \max\{p_2, t_2 + te_2\}, S_{2n}^r + t_2]$.
- Move the robot empty from M_2 to M_1 during time interval $[p_1 + t_1, p_1 + t_1 + te_1]$.
- Move the robot empty from M_3 to M_1 during time interval $[C_{2j}^r, C_{2j}^r + \sum_{i=1}^{2} te_i]$ for $j = 1, ..., n-2$.
- Move the robot empty from M_3 to M_2 during time interval $[C_{2,n-1}^r, C_{2,n-1}^r + te_2]$.

Schedule 3.1.2.2 is illustrated in Figure 16 below for $n = 4$ with empty return moves shown in bold.

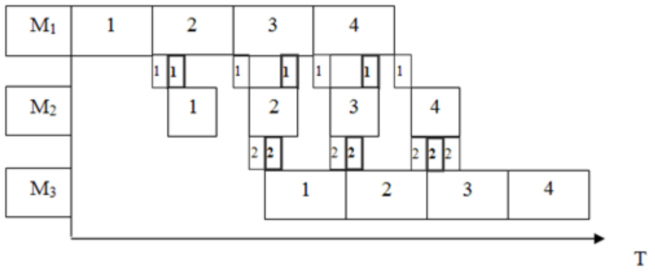

Figure 16: Schedule 3.1.2.2.

Lemma 16 *Schedule 3.1.2.2 is an optimal schedule for subcase 3.1.2.2.*

Proof. We start by showing that schedule 3.1.2.2 is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$ implies that there is no overlap between processing operations on M_1. Moreover, the fact that $S^m_{22} = 2p_1 + t_1 \geq C^m_{21} = p_1 + t_1 + p_2$, where the inequality follows from the case condition that $p_1 = \max\{p_1, t_1 + te_1, p_2\} \geq p_2$, implies that there is no overlap between the processing of J_1 and J_2 on M_2. In order to prove that such an overlap also does not exist between the processing of any other consecutive jobs on M_2, we further need to prove that

$$S^m_{2,j+1} = \max\left\{2p_1 + (j-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), (j+1)p_1\right\} + t_1 \geq$$

$$C^m_{2j} = \max\left\{2p_1 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), jp_1\right\} + t_1 + p_2$$

for $j = 2, ..., n-1$. We prove this by showing that the inequality holds for both cases when either $\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i \geq p_1$ or $\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i < p_1$. If $\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i \geq p_1$, then we need to prove that

$$2p_1 + (j-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 \geq 2p_1 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + p_2,$$

which follows from the fact that $\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i \geq p_1 \geq p_2$. If, on the other hand, $\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i < p_1$, then we need to prove that $(j+1)p_1 + t_1 \geq jp_1 + t_1 + p_2$, which holds as well due to the fact that $p_1 \geq p_2$. The fact that $S^m_{3,j+1} = 2p_1 + t_1 + t_2 + jp_3 = C^m_{3j}$ for $j = 1, ..., n$ implies that there is no overlap between processing operations on M_3 as well.

The fact that $C^m_{1j} = jp_1 = S^r_{1j}$ for $j = 1, 2$ and that $C^m_{1j} = jp_1 \leq S^r_{1j} = \max\{S^m_{2,j-1} + t_2 + \sum_{i=1}^{2} te_i, jp_1\}$ for $j = 3, ..., n$, implies that there is no overlap between the processing operation of job J_j on M_1 and the move of J_j from M_1 to M_2 for $j = 1, ..., n$. Moreover, the fact that $C^r_{1j} = jp_1 + t_1 = S^m_{2j}$ for $j = 1, 2$, implies that there is no overlap between move J_j from M_1 to M_2 and processing operations of job J_j on M_2 for $j = 1, 2$. In order to verify that such an overlap also does not exist for jobs $J_3, ..., J_n$ we have to prove that

$$C^r_{1j} = \max\{\max\left\{2p_1 + (j-3)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), (j-1)p_1\right\} + t_1 + t_2 + \sum_{i=1}^{2} te_i, jp_1\} + t_1$$

$$\leq S^m_{2j} = \max\left\{2p_1 + (j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), jp_1\right\} + t_1.$$

It is easy to show that if $\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i \geq p_1$ then $C^r_{1j} = S^m_{2j} = 2p_1 +$ $(j-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1$ and if $\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i < p_1$ then $C^r_{1j} = S^m_{2j} = jp_1 + t_1$. Thus, there is no overlap between the move of J_j from M_1 to M_2 and processing operations of job J_j on M_2 for $j = 1, ..., n$. For $j = 1, ..., n-1$, the fact that $S^r_{2j} = S^m_{2,j+1} \geq C^m_{2j}$ follows from the fact that $S^r_{2j} = S^m_{2,j+1}$ and that we have already proved above that $S^m_{2,j+1} \geq C^m_{2j}$. Moreover, we also have that $S^r_{2n} = S^m_{2n} + \max\{p_2, t_2 + te_2\} \geq C^m_{2n} = S^m_{2n} + p_2$. Thus, there is no overlap between processing operation of job J_j on M_2 and the move of J_j from M_2 to M_3 for $j = 1, ..., n$. For $j = 1, ..., n-1$ we have that

$$C^r_{2j} = S^m_{2,j+1} + t_2 = \max\left\{2p_1 + t_1 + (j-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), (j+1)p_1 + t_1\right\} + t_2 \leq$$

$$S^m_{3j} = 2p_1 + t_1 + t_2 + (j-1)p_3,$$

due to the fact that $p_3 \geq p_1$ and $p_3 \geq \left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$. This implies that there is no overlap between move J_j from M_2 to M_3 and the processing operation of job J_j on M_3 for $j = 1, ..., n-1$. In order to prove that such an overlap also does not exist for job J_n we need to prove that

$$C^r_{2n} = S^m_{2n} + \max\{p_2, t_2 + te_2\} + t_2 =$$

$$\max\left\{2p_1 + (n-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), np_1\right\} + t_1 + \max\{p_2, t_2 + te_2\} + t_2$$

$$\leq S^m_{3n} = 2p_1 + t_1 + t_2 + (n-1)p_3.$$

Consider first the case where $p_1 \geq \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$. For this case we need to prove that

$np_1+t_1+\max\{p_2, t_2+te_2\}+t_2 \leq 2p_1+t_1+t_2+(n-1)p_3$, i.e., that $(n-2)p_1+\max\{p_2, t_2+te_2\} \leq (n-1)p_3$. The last inequality holds due the fact that $\max\{p_2, t_2+te_2\} \leq p_1 \leq p_3$. Consider now the case where $p_1 < \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$. For this case we need to prove that

$$2p_1 + (n-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + \max\{p_2, t_2+te_2\} + t_2 \leq 2p_1 + t_1 + t_2 + (n-1)p_3,$$

i.e., that $(n-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + \max\{p_2, t_2+te_2\} \leq (n-1)p_3$, which directly follows from the fact that $p_3 \geq \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ and that $p_3 \geq p_2$. This completes our proof that there is no overlap between processing operations and processing and transferring operations and thus Condition 1 holds.

The robot starts its moves by transferring J_1 from M_1 to M_2 during time interval $[p_1, p_1 + t_1]$. Then the robot returns empty to M_1 during time interval $[p_1 + t_1, p_1 + t_1 + te_1]$, and performs move J_2 from M_1 to M_2 during time interval $[2p_1, 2p_1+t_1]$. Since $2p_1 \geq p_1+t_1+te_1$, there is no overlap between these two last operations. Then, for $j = 1, ..., n-1$, the robot performs move J_j from M_2 to M_3 during time interval $[S^m_{2,j+1}, S^m_{2,j+1} + t_2]$, returns empty to M_1 during time interval $[S^m_{2,j+1} + t_2, S^m_{2,j+1} + t_2 + \sum_{i=1}^{2} te_i]$, and moves J_{j+2} from M_1 to M_2 during time interval

$$[\max\{S^m_{2,j+1} + t_2 + \sum_{i=1}^{2} te_i, (j+2)p_1\}, \max\{S^m_{2,j+1} + t_2 + \sum_{i=1}^{2} te_i, (j+2)p_1\} + t_1].$$

The fact that $\max\{S^m_{2,j+1} + t_2 + \sum_{i=1}^{2} te_i, (j+2)p_1\} \geq S^m_{2,j+1} + t_2 + \sum_{i=1}^{2} te_i$ implies that there is no overlap between these operations as well. Lastly, after performing move J_n from M_1 to M_2 during time interval

$$[S^r_{1n}, C^r_{1n}] = [\max\{S^m_{2,n-1} + t_2 + \sum_{i=1}^{2} te_i, np_1\}, \max\{S^m_{2,n-1} + t_2 + \sum_{i=1}^{2} te_i, np_1\} + t_1],$$

the robot moves J_{n-1} from M_2 to M_3 during time interval $[S^r_{2,n-1}, C^r_{2,n-1}] = [S^m_{2n}, S^m_{2n} + t_2]$, returns empty to M_2 during time interval $[C^r_{2,n-1}, C^r_{2,n-1} + te_2] = [S^m_{2n} + t_2, S^m_{2n} + t_2 + te_2]$, and finishes its operations by performing move J_n from M_2 to M_3 during time interval $[S^m_{2n}+\max\{p_2, t_2+te_2\}, S^m_{2n}+\max\{p_2, t_2+te_2\}+t_2]$ (note that since $\max\{p_2, t_2+te_2\} \geq t_2+te_2$ there is no overlap between these last two operations). In order to complete the proof that there is no overlap between robot operations, we still need to show that there is no overlap between operations J_n from M_1 to M_2 and J_{n-1} from M_2 to M_3, i.e., we need to prove that

$$S^r_{2,n-1} = S^m_{2n} = \max\left\{2p_1 + (n-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), np_1\right\} + t_1 \geq C^r_{1n} =$$

$$\max\{\max\left\{2p_1 + (n-3)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), (n-1)p_1\right\} + t_1 + t_2 + \sum_{i=1}^{2} te_i, np_1\} + t_1.$$

The fact that $S^r_{2,n-1} = C^r_{1n} = 2p_1 + (n-2)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1$ when $\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i \geq p_1$ and $S^r_{2,n-1} = C^r_{1n} = np_1 + t_1$ when $\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i < p_1$ completes this proof, and thus Condition 2 holds.

The fact that $C^m_{1j} = jp_1 = S^r_{1j}$ for $j = 1, 2$ and that $C^m_{1j} = jp_1 \leq S^r_{1j} = \max\{S^m_{2,j-1} + t_2 + \sum_{i=1}^{2} te_i, jp_1\}$ for $j = 3, ..., n$ implies that job J_j is ready to move from M_1 to M_2 at time S^r_{1j}. Moreover, the fact that $C^m_{2j} \leq S^r_{2j} = S^m_{2,j+1}$ for $j = 1, ..., n-1$ and $C^m_{2n} = S^m_{2n} + p_2 \leq S^r_{2n} = S^m_{2n} + \max\{p_2, t_2 + te_2\}$ implies that job J_j is ready to move from M_2 to M_3 at time S^r_{2j}. Thus, Condition 3 holds. ■

It is implied from the feasibility of schedule 3.2.2.2 that the completion time of job J_n on machine M_3 is at time $C^m_{3n} = 2p_1 + t_1 + t_2 + np_3$. The fact that this time matches the lower bound in (eq. 80) when

$$p_1 - p_2 \leq \min\{(n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3\right), \Delta(x^*)\}$$

implies that this schedule 3.1.2.2 is optimal for subcase 3.1.2.2.

Optimal schedule for subcase 3.1.2.3 For subcase 3.1.2.3, where $p_3 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ and $p_1 = \max\{p_1, t_1 + te_1, p_2\}$ and

$$\min\left\{(n-1)\left(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3\right), p_1 - p_2, \Delta(x^*)\right\} = \Delta(x^*)$$

, we define the following schedule (Schedule 3.1.2.3). In general, given $x \in \{2, ..., n-1\}$, we show below that we can construct a feasible schedule (Schedule 3.1.2.3) with a makespan value of $LB_3'''' = LB_3 + \Delta(x)$. This implies that for x^* the schedule is feasible and thus optimal as well.

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S^m_{1j}, C^m_{1j}] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.

- Schedule job J_j on M_2 during time interval $[S^m_{2j}, C^m_{2j}] = [p_1 + t_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), S^m_{2j} + p_2]$ for $j = 1, ..., x$.

- Schedule job J_{x+1} on M_2 during time interval $[S^m_{2,x+1}, C^m_{2,x+1}] = [\max\{\max\{S^m_{2x} + te_1, (x+1)p_1\} + t_1, S^m_{2x} + p_2\}, S^m_{2,x+1} + p_2]$.

- Schedule job J_j on M_2 during time interval $[S^m_{2j}, C^m_{2j}] = [\max\{\max\{S^m_{2,j-1} + t_2 + \sum_{i=1}^{2} te_i, jp_1\} + t_1, C^m_{2,j-1}\}, S^m_{2j} + p_2]$ for $j = x+2, ..., n$.

- Schedule job J_j on M_3 during time interval $[S^m_{3j}, C^m_{3j}] = [p_1 + t_1 + p_2 + t_2 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i), S^m_{3j} + p_3]$ for $j = 1, ..., x-1$.

- Schedule job J_x on M_3 during time interval $[S^m_{3x}, C^m_{3x}] = [S^m_{2,x+1} + t_2, S^m_{3x} + p_3]$.

- Schedule job J_j on M_3 during time interval $[S^m_{3j}, C^m_{3j}] = [S^m_{3x} + (j-x)p_3, S^m_{3x} + (j-x+1)p_3]$ for $j = x+1, ..., n$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $[S^r_{1j}, C^r_{1j}] = [p_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), S^r_{1j} + t_1]$ for $j = 1, ..., x$.

- Move job J_{x+1} from M_1 to M_2 during time interval $[S^r_{1,x+1}, C^r_{1,x+1}] = [\max\{S^m_{2x} + te_1, (x+1)p_1\}, S^r_{1,x+1} + t_1]$.

- Move job J_j from M_1 to M_2 during time interval $[S^r_{1j}, C^r_{1j}] = [\max\{S^m_{2,j-1} + t_2 + \sum_{i=1}^{2} te_i, jp_1\}, S^r_{1j} + t_1]$ for $j = x+2, ..., n$.

- Move job J_j from M_2 to M_3 during time interval $[S^r_{2j}, C^r_{2j}] = [C^m_{2j}, S^r_{2j} + t_2]$ for $j = 1, ..., x-1$.

- Move job J_j from M_2 to M_3 during time interval $[S^r_{2j}, C^r_{2j}] = [S^m_{2,j+1}, S^r_{2j} + t_2]$ for $j = x, ..., n-1$.

- Move job J_n from M_2 to M_3 during time interval $[S^r_{2n}, C^r_{2n}] = [S^m_{2n} + \max\{t_2 + te_2, p_2\}, S^r_{2n} + t_2]$.

- Move the robot empty from M_3 to M_1 during time interval $[C^r_{2j}, C^r_{2j} + \sum_{i=1}^{2} te_i]$ for $j = 1, ..., n-1$.

- Move the robot empty from M_2 to M_1 during time interval $[S^m_{2x}, S^m_{2x} + te_1]$.

- Move the robot empty from M_3 to M_2 during time interval $[C^r_{2,n-1}, C^r_{2,n-1} + te_2]$.

Schedule 3.1.2.3 is illustrated in Figure 17 below for $n = 4$ and $x = 2$ with empty return moves shown in bold.

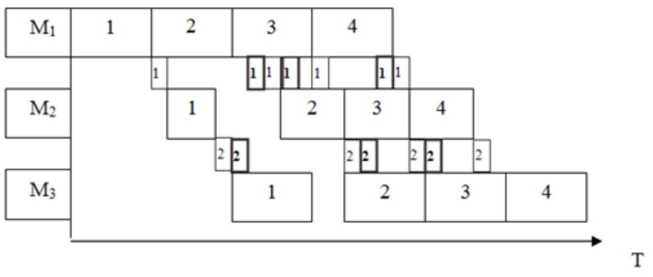

Figure 17: Schedule 3.2.2.3.

Lemma 17 *Schedule 3.1.2.3 is an optimal schedule for subcase 3.1.2.3.*

Proof. We start by showing that schedule 3.1.2.3 is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$ implies that there is no overlap between processing operations on M_1. Moreover, the fact that

$$S^m_{2,j+1} = p_1+t_1+j\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right) \geq C^m_{2j} = p_1+t_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2$$

for $j = 1, ..., x-1$; that

$$S^m_{2,x+1} = \max\{\max\{S^m_{2x}+te_1,(x+1)p_1\}+t_1, S^m_{2x}+p_2\} \geq C^m_{2x} = S^m_{2x}+p_2;$$

and that

$$S^m_{2,j+1} = \max\left\{\max\left\{S^m_{2j}+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i,(j+1)p_1+t_1\right\}, C^m_{2j}\right\} \geq C^m_{2j}$$

for $j = x+1, ..., n-1$, implies that there is no overlap between the processing of jobs on M_2. Moreover, the fact that

$$\begin{aligned} S^m_{3,j+1} &= \sum_{i=1}^{2} p_i+\sum_{i=1}^{2} t_i+j(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i) > \\ C^m_{3j} &= \sum_{i=1}^{2} p_i+\sum_{i=1}^{2} t_i+(j-1)(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i)+p_3 \end{aligned}$$

for $j = 1, ..., x-2$ (since $p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i > p_3$); that

$$S_{3x}^m = S_{2,x+1}^m + t_2 \geq S_{2x}^m + p_2 + t_2 = \sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$$

$$> C_{3,x-1}^m = \sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (x-2)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + p_3$$

(since $p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i > p_3 > p_1$); and that $S_{3j}^m = C_{3,j-1}^m = S_{3x}^m + (j-x)p_3$ for $j = x+1, ..., n$, implies that there is no overlap between processing operations on M_3 as well.

The fact that $C_{1j}^m = jp_1 < S_{1j}^r = p_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$ for $j = 1, ..., x$ (since $p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i > p_3 \geq p_1$); that $C_{1,x+1}^m = (x+1)p_1 \leq S_{1,x+1}^r = \max\{S_{2x}^m + te_1, (x+1)p_1\}$; and that $C_{1j}^m = jp_1 \leq S_{1j}^r = \max\{S_{2,j-1}^m + t_2 + \sum_{i=1}^{2} te_i, jp_1\}$ for $j = x+2, ..., n$, implies that there is no overlap between the processing operation of job J_j on M_1 and the move of J_j from M_1 to M_2 for $j = 1, ..., n$. Moreover, the fact that $C_{1j}^r = p_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 = S_{2j}^m$ for $j = 1, ..., x$; that

$$C_{1,x+1}^r = \max\{S_{2x}^m + te_1, (x+1)p_1\} + t_1 \leq S_{2,x+1}^m = \max\{\max\{S_{2x}^m + te_1, (x+1)p_1\} + t_1, S_{2x}^m + p_2\};$$

and that

$$C_{1j}^r = \max\{S_{2,j-1}^m + t_2 + \sum_{i=1}^{2} te_i, jp_1\} + t_1 \leq S_{2j}^m = \max\{\max\{S_{2,j-1}^m + t_2 + \sum_{i=1}^{2} te_i, jp_1\} + t_1, C_{2,j-1}^m\}$$

for $j = x+2, ..., n$, implies that the is no overlap between the move of J_j from M_1 to M_2 and processing operations of job J_j on M_2 for $j = 1, ..., n$. Furthermore, the fact that $S_{2j}^r = C_{2j}^m$ for $j = 1, ..., x-1$; that

$$S_{2x}^r = S_{2,x+1}^m = \max\{\max\{ S_{2x}^m + te_1, (x+1)p_1\} + t_1\}, S_{2x}^m + p_2\} \geq C_{2x}^m = S_{2x}^m + p_2;$$

that $S_{2j}^r = S_{2,j+1}^m \geq C_{2j}^m$ for $j = x+1, ..., n-1$ (as already proven above); and that $S_{2n}^r = S_{2n}^m + \max\{t_2 + te_2, p_2\} \geq C_{2n}^m = S_{2n}^m + p_2$, implies that there is no overlap between the processing operation of job J_j on M_2 and the move of J_j from M_2 to M_3 for $j = 1, ..., n$. Finally, the fact that

$$C_{2j}^r = C_{2j}^m + t_2 = p_1 + t_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + p_2 + t_2 = S_{3j}^m$$

for $j = 1, ..., x-1$ and that $C_{2x}^r = S_{2,x+1}^m + t_2 = S_{3x}^m$, implies that in order to prove that there is no overlap between the move of J_j from M_2 to M_3 for $j = 1, ..., n$ and the processing operation of job J_j on M_3, it only remains to prove that

$$C^r_{2j} = S^m_{2,j+1} + t_2 = \max\left\{\max\left\{S^m_{2j} + t_2 + \sum_{i=1}^{2} te_i, (j+1)p_1\right\} + t_1, S^m_{2j} + p_2\right\} + t_2$$
$$= \max\left\{S^m_{2j} + \max\left\{\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i, p_2\right\}, (j+1)p_1 + t_1\right\} + t_2$$
$$\leq S^m_{3j} = S^m_{3x} + (j-x)p_3 = S^m_{3x} + (j-x)p_3 = S^m_{2,x+1} + t_2 + (j-x)p_3,$$

i.e., that

$$C^r_{2j} = S^m_{2,j+1} + t_2 \leq S^m_{3x} + (j-x)p_3 = S^m_{2,x+1} + t_2 + (j-x)p_3 \tag{81}$$

for $j = x+1, ..., n-1$.

We prove that the condition in (81) holds by induction. To do so, we first prove that the condition in (81) holds for $j = x+1$. For $j = x+1$ the inequality in (81) reduces to

$$\max\left\{S^m_{2,x+1} + \max\left\{\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i, p_2\right\}, (x+2)p_1 + t_1\right\} + t_2 \leq S^m_{2,x+1} + t_2 + p_3. \tag{82}$$

Since $p_3 \geq \max\left\{\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i, p_2\right\}$, in order to prove that the condition in (82) holds, we only need to prove that

$$S^m_{2,x+1} + p_3 \geq (x+2)p_1 + t_1. \tag{83}$$

The correctness of the inequality in (83) directly follows from the fact that

$$S^m_{2,x+1} + p_3 = \max\{\max\{S^m_{2x} + te_1, (x+1)p_1\} + t_1, S^m_{2x} + p_2\} + p_3 \geq (x+1)p_1 + t_1 + p_3,$$

and from the fact that $p_3 \geq p_1$.

Now, assume that the condition in (81) holds for $j \in \{x+2, ..., n-2\}$ (the induction assumption); we need to prove that the condition in (81) holds also for $j+1$, i.e., that

$$\max\left\{S^m_{2,j+1} + Y, (j+2)p_1 + t_1\right\} + t_2 \leq S^m_{2,x+1} + t_2 + (j-x+1)p_3, \tag{84}$$

where $Y = \max\left\{\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i, p_2\right\}$. Since

$$\max\left\{S^m_{2,j+1} + Y, (j+2)p_1 + t_1 + t_2\right\} =$$

$$\max\left\{\max\left\{\max\left\{S^m_{2j} + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i, (j+1)p_1 + t_1\right\}, C^m_{2,j}\right\} + Y, (j+2)p_1 + t_1\right\} + t_2 =$$

$$\max\left\{\max\left\{\max\left\{S^m_{2j} + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i, (j+1)p_1 + t_1\right\}, S^m_{2j} + p_2\right\} + Y, (j+2)p_1 + t_1\right\} + t_2$$

$$= \max\left\{\max\left\{S^m_{2j} + Y, (j+1)p_1 + t_1\right\} + t_2 + Y, (j+2)p_1 + \sum_{i=1}^{2} t_i\right\} \leq$$

$$\max\left\{S_{2,x+1} + t_2 + (j-x)p_3 + Y, (j+2)p_1 + \sum_{i=1}^{2} t_i\right\},$$

where the last inequality follows from the induction assumption, implies that in order to prove that the condition in (84) holds, we only need to prove that

$$\max\left\{S_{2,x+1} + t_2 + (j-x)p_3 + Y, (j+2)p_1 + \sum_{i=1}^{2} t_i\right\} \leq S^m_{2,x+1} + t_2 + (j-x+1)p_3. \quad (85)$$

If

$$\max\left\{S_{2,x+1} + t_2 + (j-x)p_3 + Y, (j+2)p_1 + \sum_{i=1}^{2} t_i\right\} = S_{2,x+1} + t_2 + (j-x)p_3 + Y$$

then the correctness of (85) follows from the fact that $p_3 \geq Y = \max\left\{\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i, p_2\right\}$. If, however,

$$\max\left\{S_{2,x+1} + t_2 + (j-x)p_3 + Y, (j+2)p_1 + \sum_{i=1}^{2} t_i\right\} = (j+2)p_1 + \sum_{i=1}^{2} t_i$$

, then we need to prove that for

$S^m_{2,x+1} + t_2 + (j-x+1)p_3 = \max\{\max\{S^m_{2x} + te_1, (x+1)p_1\} + t_1, S^m_{2x} + p_2\} + t_2 + (j-x+1)p_3$

$$(j+2)p_1 + \sum_{i=1}^{2} t_i \leq S^m_{2,x+1} + t_2 + (j-x+1)p_3. \quad (86)$$

The fact that

$$\max\{\max\{S^m_{2x}+te_1,(x+1)p_1\}+t_1,S^m_{2x}+p_2\}+t_2+(j-x+1)p_3 \geq (x+1)p_1+\sum_{i=1}^{2} t_i+(j-x+1)p_3$$

implies that in order to prove that the condition in (86) holds, we only need to prove that $(x+1)p_1+\sum_{i=1}^2 t_i+(j-x+1)p_3 \geq (j+2)p_1+\sum_{i=1}^2 t_i$. The fact that the last inequality holds directly follows from the fact that $p_3 \geq p_1$ and completes the proof that there is no overlap between the move of J_j from M_2 to M_3 for $j = 1, ..., n$ and the processing operation of job J_j on M_3. Thus Condition 1 holds.

The robot starts its moves by moving J_j from M_1 to M_2 for $j = 1, ..., x-1$, during time interval

$$[p_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right), p_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+t_1].$$

Then, after waiting p_2 units of time beside M_2 it performs move J_j from M_2 to M_3 during time interval

$$[p_1+t_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2, p_1+t_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2+t_2].$$

Then, the robot returns empty to M_1 during time interval

$$[p_1+t_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2+t_2,$$
$$p_1+t_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2+t_2+\sum_{i=1}^{2} te_i]$$

in order to perform J_{j+1} from M_1 to M_2 during time interval

$$[p_1+j\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right), p_1+j\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+t_1].$$

The fact that

$$p_1+j\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)=p_1+t_1+(j-1)\left(p_2+\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i\right)+p_2+t_2+\sum_{i=1}^{2} te_i$$

implies that there is no overlap between these two last operations.

After completing the move of J_x from M_1 to M_2 at time $p_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1$, the robot returns empty to M_1 during time interval

$$[S_{2x}^m, S_{2x}^m + te_1] =$$
$$[p_1 + t_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right), p_1 + t_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + te_1],$$

and then performs move J_{x+1} from M_1 to M_2 during time interval $[\max\{S_{2x}^m + te_1, (x+1)p_1\}, \max\{S_{2x}^m + te_1, (x+1)p_1\} + t_1]$. This set of moves is followed by move J_x from M_2 to M_3, which is done during time interval $[S_{2,x+1}^m, S_{2x}^r + t_2]$. Since

$$S_{2,x+1}^m = \max\{\max\{S_{2x}^m + te_1, (x+1)p_1\} + t_1\}, S_{2x}^m + p_2\} \geq \max\{S_{2x}^m + te_1, (x+1)p_1\} + t_1,$$

there is no overlap between moves J_{x+1} from M_1 to M_2 and J_x from M_2 to M_3. After completing the move of J_x from M_2 to M_3 at time $C_{2x}^r = \max\{\max\{S_{2x}^m + te_1, (x+1)p_1\} + t_1\}, S_{2x}^m + p_2\} + t_2$, the robot returns empty to M_1 during time interval $[C_{2x}^r, C_{2x}^r + \sum_{i=1}^{2} te_i] = [S_{2x}^r + t_2, S_{2x}^r + t_2 + \sum_{i=1}^{2} te_i]$. Then, for $j = x+2, ..., n-1$, the robot move J_j from M_1 to M_2, during time interval $[C_{1j}^r, C_{1j}^r + t_1] = [\max\{S_{2,j-1}^m + t_2 + \sum_{i=1}^{2} te_i, jp_1\}, \max\{S_{2,j-1}^m + t_2 + \sum_{i=1}^{2} te_i, jp_1\} + t_1]$, followed by the move of J_{j-1} from M_2 to M_3 during time interval $[S_{2j}^m, S_{2j}^m + t_2]$. The fact that $S_{2x}^r = S_{2,x+1}^m$ implies that the robot returns empty to M_1 at time $S_{2,x+1}^m + t_2 + \sum_{i=1}^{2} te_i \leq \max\{S_{2,x+1}^m + t_2 + \sum_{i=1}^{2} te_i, (x+2)p_1\} = C_{1,x+2}^r$, which implies that there is no overlap between the operation of returning to M_1 and the move of J_{x+2} from M_1 to M_2. Moreover, the fact that

$$\max\{S_{2,j-1}^m + t_2 + \sum_{i=1}^{2} te_i, jp_1\} + t_1 \leq S_{2j}^m = \max\left\{\max\{S_{2,j-1}^m + t_2 + \sum_{i=1}^{2} te_i, jp_1\} + t_1, C_{2,j-1}^m\right\}$$

implies that there is no overlap between the moves of J_j from M_1 to M_2 and J_{j-1} from M_2 to M_3. After completing move J_{j-1} from M_2 to M_3 at time $S_{2j}^m + t_2$, the robot returns empty to M_1 during time interval $[S_{2j}^m + t_2, S_{2j}^m + t_2 + \sum_{i=1}^{2} te_i]$ in order to move J_{j+1} from M_1 to M_2 during time interval

$$[\max\{S_{2,j}^m + t_2 + \sum_{i=1}^{2} te_i, (j+1)p_1\}, \max\{S_{2,j}^m + t_2 + \sum_{i=1}^{2} te_i, (j+1)p_1\} + t_1].$$

Lastly, after completing the move of J_n from M_1 to M_2 at time $\max\{S_{2,n-1}^m + t_2 + \sum_{i=1}^{2} te_i, np_1\} + t_1$, the robot performs move J_{n-1} from M_2 to M_3 during time interval $[S_{2n}^m, S_{2n}^m + t_2]$, where $S_{2n}^m = \max\{\max\{S_{2,n-1}^m + t_2 + \sum_{i=1}^{2} te_i, np_1\} + t_1, C_{2,n-1}^m\}$, and returns empty to M_2 during time interval $[C^r_{2,n-1}, C^r_{2,n-1}+te_2] = [S^m_{2n}+t_2, S^m_{2n}+t_2+te_2]$ to move J_n from M_2 to M_3 during time interval $[S^m_{2n}+\max\{t_2+te_2, p_2\}, S^m_{2n}+\max\{t_2+te_2,p_2\}+t_2]$. The fact that $\max\{S^m_{2,n-1}+t_2+\sum_{i=1}^2 te_i, np_1\}+t_1 = S^m_{2n}$ and that $S^m_{2,n}+t_2+te_2 \le S^m_{2n}+\max\{t_2+te_2,p_2\}$ implies that there is no overlap between these last operations as well and thus Condition 2 holds.

The fact that $C^m_{1j} = jp_1 < S^r_{1j} = p_1+(j-1)\left(p_2+\sum_{i=1}^2 t_i+\sum_{i=1}^2 te_i\right)$ for $j=1,...,x$ (since $p_2+\sum_{i=1}^2 t_i+\sum_{i=1}^2 te_i > p_3 > p_1$); that $C^m_{1,x+1} = (x+1)p_1 \le S^r_{1,x+1} = \max\{S^m_{2x}+te_1, (x+1)p_1\}$; and that $C^m_{1j} = jp_1 \le S^r_{1j} = \max\{S^m_{2,j-1}+t_2+\sum_{i=1}^2 te_i, jp_1\}$ for $j = x+2,...,n$ implies that job J_j $j=1,...,n$ is ready to move from M_1 to M_2 at time S^r_{1j}. Moreover, the fact that $C^m_{2j} = S^r_{2j}$ for $j=1,...,x-1$; that

$$C^m_{2x} = S^m_{2x}+p_2 \le S^r_{2x} = S^m_{2,x+1} = \max\{\max\{S^m_{2x}+te_1, (x+1)p_1\}+t_1\}, S^m_{2x}+p_2\};$$

that

$$C^m_{2,x+1} \le S^r_{2,x+1} = S^m_{2,x+2} = \max\{\max\{S^m_{2,x+1}+t_2+\sum_{i=1}^2 te_i, (x+2)p_1\}+t_1, C^m_{2,x+1}\};$$

that $C^m_{2j} \le S^r_{2j} = S^m_{2,j+1}$ for $j=x+2,...,n-1$ (as proven above); and that $C^m_{2n} = S^m_{2n}+p_2 \le S^r_{2n} = S^m_{2n}+\max\{p_2, t_2+te_2\}$, implies that job J_j for $j=1,...,n$ is ready to move from M_2 to M_3 at time S^r_{2j}. Thus, Condition 3 holds.

It is implied from the feasibility of schedule 3.1.2.3 that the completion time of job J_n on machine M_3 is at time ($\hat{Y} = (n-x+1)p_3$)

$$\begin{gathered}C^m_{3n} = S^m_{3x}+\hat{Y} = \max\{\max\{S^m_{2x}+te_1, (x+1)p_1\}+t_1, S^m_{2x}+p_2\}+t_2+\hat{Y} = \\ \max\{S^m_{2x}+\max\{te_1+t_1, p_2\}, (x+1)p_1+t_1\}+t_2+\hat{Y} = \\ \max\left\{p_1+t_1+(x-1)\left(p_2+\sum_{i=1}^2 t_i+\sum_{i=1}^2 te_i\right)+\max\{te_1+t_1,p_2\}, (x+1)p_1+t_1\right\}+t_2+\hat{Y}.\end{gathered}$$

The fact that this time matches the lower bound in (eq. 80) for x^* when

$$\Delta(x^*) \le \min\{(n-1)\left(\textstyle\sum_{i=1}^2 t_i+\sum_{i=1}^2 te_i+p_2-p_3\right), p_1-p_2\}$$

implies that this schedule 3.1.2.3 is optimal for subcase 3.1.2.3. ■

The analysis of subcase 3.1.3

In subcase 3.1.3, where $p_3 < p_2+\sum_{i=1}^2 t_i+\sum_{i=1}^2 te_i$ and $t_1+te_1 = \max\{p_1, t_1+te_1, p_2\}$, we first update LB_3 and provide a revised (tighter) bound denoted by LB_3''''. We consider two possible scenarios: (a) the robot does not perform two consecutive moves from M_1 to M_2, and (b) at least once the robot makes two consecutive moves from M_1 to M_2. Let us first consider case (a). In this case the robot moves each job from M_1 to M_2 and then to M_3 before moving to the next job, i.e., after transferring job J_j $(j = 1, ..., n)$ from M_1 to M_2, the robot waits beside M_2 for the completion of J_j on this machine and then moves it to M_3. It thus implies that the robot will start the move of J_1 from M_1 to M_2 not earlier than p_1, and will start the move of J_j from M_1 to M_2 not earlier than $p_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$ for $j = 2, ..., n$. Thus, job J_n will not finish its processing on M_3 earlier than time

$$LB_3''''(a) = p_1 + (n-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + p_2 + t_2 + p_3 \geq p_1 + t_1 + p_2 + t_2 + np_3 = LB_3.$$

Consider the next case (b) where the robot at least once makes two consecutive moves from M_1 to M_2. We further divide case (b) to two subcases. The first $(b1)$, where the first two consecutive moves from M_1 to M_2 are done on jobs J_1 and J_2, and the second $(b2)$, when it is done on jobs J_x and J_{x+1} with $2 \leq x \leq n-1$. In subcase $(b1)$, the robot will start the move of J_2 from M_1 to M_2 not earlier than at time $p_1 + t_1 + te_1$, which further implies that the move of J_1 from M_2 to M_3 will not start before $p_1 + t_1 + te_1 + t_1 = p_1 + 2t_1 + te_1$. Thus, job J_1 will not start its processing on M_3 before $p_1 + 2t_1 + te_1 + t_2$, and the makespan value will be not less than

$$LB_3''''(b1) = p_1 + 2t_1 + te_1 + t_2 + np_3 \geq p_1 + t_1 + p_2 + t_2 + np_3 = LB_3.$$

In subcase $(b2)$, the move of J_{x-1} from M_2 to M_3 will not finish before time $p_1 + (x-2)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + p_2 + t_2$. Thus, the move of J_x from M_1 to M_2 will not start before $p_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$, which further implies that the move of J_{x+1} from M_1 to M_2 will not start before $p_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + te_1$. Thus, the move of J_x from M_2 to M_3 will not finish before $p_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + te_1 + t_1 + t_2$, and the makespan value in this case is lower bounded by

$$p_1 + (x-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + te_1 + t_1 + t_2 + (n-x+1)p_3.$$

Since $p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i > p_3$ and $x \geq 2$, this value is lower bounded by

$$LB_3''''(b1) = p_1 + \left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + te_1 + t_1 + t_2 + (n-1)p_3 \geq$$
$$p_1 + t_1 + te_1 + t_1 + t_2 + np_3 = LB_3''''(b1).$$

To conclude our analysis, we have that if scenario (a) is selected, then

$$C_{\max} \geq p_1 + (n-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + p_2 + t_2 + p_3,$$

and if scenario (b) is selected, then $C_{\max} \geq p_1 + 2t_1 + te_1 + t_2 + np_3 = LB_3''''(b1)$. Thus, the makespan value will not be less than

$$\min\left\{\sum_{i=1}^{3} p_i + (n-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + \sum_{i=1}^{2} t_i, p_1 + 2t_1 + te_1 + t_2 + np_3\right\} =$$

$$\sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + np_3 + \min\left\{t_1 + te_1 - p_2, (n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3)\right\},$$

and we have that

$$LB_3''' = LB_3 + \min\left\{(t_1 + te_1) - p_2, (n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3)\right\} \tag{87}$$

Below, we provide an optimal schedule that matches the lower bound value in (87) for each of the following two scenarios that can arise. The first subcase 3.1.3.1 is where $(t_1 + te_1) - p_2 \leq (n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3)$, the second subcase 3.1.3.2 is the opposite case.

Optimal schedule for subcase 3.1.3.1 For subcase 3.1.3.1, where $p_3 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$, $t_1 + te_1 = \max\{p_1, t_1 + te_1, p_2\}$ and

$$(t_1 + te_1) - p_2 = \min\left\{(t_1 + te_1) - p_2, (n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3)\right\},$$

we define the following schedule (Schedule 3.1.3.1):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S_{1j}^m, C_{1j}^m] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.

- Schedule job J_1 on M_2 during time interval $[S_{21}^m, C_{21}^m] = [p_1 + t_1, p_1 + t_1 + p_2]$.
- Schedule job J_j on M_2 during time interval $[S_{2j}^m, C_{2j}^m] = [p_1 + 2t_1 + te_1 + (j-2)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i), S_{2j}^m + p_2]$ for $j = 2, ..., n$.
- Schedule job J_1 on M_3 during time interval $[S_{31}^m, C_{31}^m] = [p_1 + 2t_1 + t_2 + te_1, S_{3j}^m + p_3]$.
- Schedule job J_j on M_3 during time interval $[S_{3j}^m, C_{3j}^m] = [C_{3,j-1}^m, S_{3j}^m + p_3]$ for $j = 2, ..., n$.

Robot Schedule:

- Move job J_1 from M_1 to M_2 during time interval $[S_{11}^r, C_{11}^r] = [p_1, p_1 + t_1]$.
- Move job J_j from M_1 to M_2 during time interval $[S_{1j}^r, C_{1j}^r] = [p_1 + t_1 + te_1 + (j - 2)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i), S_{1j}^r + t_1]$ for $j = 2, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $[S_{2j}^r, C_{2j}^r] = [p_1 + 2t_1 + te_1 + (j - 1)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i), S_{2j}^r + t_2]$ for $j = 1, ..., n - 1$.
- Move job J_n from M_2 to M_3 during time interval $[S_{2n}^r, C_{2n}^r] = [\max\{C_{2,n-1}^r + te_2, C_{2n}^m\}, S_{2n}^r + t_2]$.
- Move the robot empty from M_2 to M_1 during time interval $[p_1 + t_1, p_1 + t_1 + te_1]$.
- Move the robot empty from M_3 to M_1 during time interval $[p_1 + 2t_1 + te_1 + (j-1)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i) + t_2, p_1 + 2t_1 + te_1 + (j-1)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i) + t_2 + \sum_{i=1}^2 te_i]$ for $j = 1, ..., n - 2$.
- Move the robot empty from M_3 to M_2 during time interval $[p_1 + 2t_1 + te_1 + (n-2)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i) + t_2, p_1 + 2t_1 + te_1 + (n-2)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i) + t_2 + te_2]$.

Schedule 3.1.3.1 is illustrated in Figure 18 below for $n = 4$ with empty return moves shown in bold.

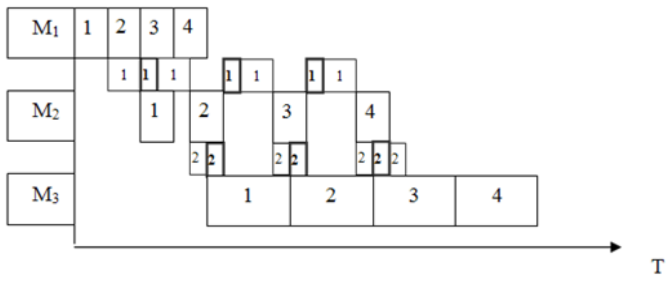

Figure 18: Schedule 3.1.3.1.

Lemma 18 *Schedule 3.1.3.1 is an optimal schedule for subcase 3.1.3.1.*

Proof. We start by showing that Schedule 3.1.3.1 is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$; that $S^m_{22} = p_1 + 2t_1 + te_1 \geq C^m_{21} = p_1 + t_1 + p_2$; that

$$S^m_{2,j+1} = p_1+2t_1+te_1+(j-1)(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i) \geq C^m_{2j} = p_1+2t_1+te_1+(j-2)(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i)+p_2$$

for $j = 2, ..., n-1$; and that $S^m_{3j} = C^m_{3,j-1}$ for $j = 2, ..., n$, where the first inequality follows from the fact that $t_1 + te_1 \geq p_2$, and the second inequality follows from the fact that $\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i \geq t_1 + te_1 \geq p_2$, implies that there is no overlap between the processing of different jobs on each of the machines. Moreover, the fact that $C^m_{11} = S^r_{11} = p_1$, that $C^m_{1j} = jp_1 \leq S^r_{1j} = p_1+t_1+te_1+(j-2)(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i)$ for $j = 2, ..., n$ (since $t_1+te_1 \geq p_1$); that $C^r_{11} = S^m_{21} = p_1 + t_1$, that $C^r_{1j} = S^m_{2j} = p_1 + t_1 + te_1 + (j-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i)$ for $j = 2, ..., n$; that $C^m_{21} = p_1 + t_1 + p_2 \leq S^r_{21} = p_1 + 2t_1 + te_1$ (since $t_1 + te_1 \geq p_2$); that

$$C^m_{2j} = p_1+2t_1+te_1+(j-2)(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i)+p_2 < S^r_{2j} = p_1+2t_1+te_1+(j-1)(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i)$$

for $j = 2, ..., n-1$ (since $\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i > t_1 + te_1 \geq p_2$); that

$$\begin{aligned} C^m_{2n} = p_1 + 2t_1 + te_1 + (n-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + p_2 \leq S^r_{2n} \\ = \max\{p_1 + 2t_1 + te_1 + (n-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + t_2 + te_2, C^m_{2n}\}; \end{aligned}$$

that

$$C^r_{2j} = p_1+2t_1+te_1+(j-1)(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i)+t_2 \leq S^m_{3j} = C^m_{3,j-1} = p_1+2t_1+te_1+t_2+(j-1)p_3$$

for $j = 1, ..., n-1$; and that

$$\begin{aligned} C^r_{2n} &= \max\{C^r_{2,n-1} + te_2, C^m_{2n}\} + t_2 = \\ &\max\{C^r_{2,n-1} + te_2, p_1 + 2t_1 + te_1 + (n-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + p_2\} \\ &\leq S^m_{3n} = p_1 + 2t_1 + te_1 + t_2 + (n-1)p_3 \end{aligned}$$

(since $p_3 \geq p_2$ and $p_3 \geq \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i$), implies that there is no overlap between processing and transferring operations of job J_j for $j = 1, ..., n$. Thus, Condition 1 holds.

Once job J_1 is completed on M_1, the robot moves this job to M_1 during time interval $[p_1, p_1 + t_1]$ and returns empty to M_1 during time interval $[p_1 + t_1, p_1 + t_1 + te_1]$. Then, for $j = 2, ..., n$, the robot performs move J_j from M_1 to M_2 followed by move job J_{j-1} from M_2 to M_3 and returns back to machine M_1 if $j < n-1$ and M_2 otherwise. The fact that $C^r_{1j} = S^r_{2,j-1} = p_1 + t_1 + te_1 + (j-2)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i) + t_1$ for $j = 2, ..., n$, and that for $j < n-1$ the robot returns to machine M_1 after completing move J_{j-1} from M_2 to M_3 at time

$$p_1+2t_1+te_1+(j-2)(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i)+t_2+\sum_{i=1}^{2} te_i = S^r_{1,j+1} = p_1+2t_1+te_1+(j-1)(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i),$$

implies that there is no overlap between robot operations. Thus, Condition 2 holds.

The fact that $C^m_{11} = S^r_{11} = p_1$, and that $C^m_{1j} = jp_1 \leq S^r_{1j} = p_1 + t_1 + te_1 + (j-2)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i)$ for $j = 2, ..., n$ (since $\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i \geq t_1 + te_1 \geq p_1$) implies that job J_j is ready to move from M_1 to M_2 at time S^r_{1j} for $j = 1, ..., n$. Moreover, the fact that $C^m_{21} = p_1 + t_1 + p_2 \leq S^r_{21} = p_1 + 2t_1 + te_1$ (since $t_1 + te_1 \geq p_2$); that

$$C^m_{2j} = p_1+2t_1+te_1+(j-2)(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i)+p_2 \leq S^r_{2j} = p_1+2t_1+te_1+(j-1)(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i)$$

for $j = 2, ..., n-1$ (since $\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i \geq t_1 + te_1 \geq p_2$); and that $C^m_{2n} \leq S^r_{2n} = \max\{C^r_{2,n-1} + te_2, C^m_{2n}\}$, implies that job J_j is ready to move from M_2 to M_3 at time S^r_{2j} for $j = 1, ..., n$. Thus, Condition 3 holds as well.

The fact that the completion time of J_n on M_3 in schedule 3.1.3.1 is at time $p_1 + 2t_1 + t_2 + te_1 + np_3$, which matches the lower bound in (eq. 87) for this subcase when $(t_1 + te_1) - p_2 \leq (n-1)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2 - p_3)$, implies that schedule 3.1.3.1 is optimal for subcase 3.1.3.1. ■

Optimal schedule for subcase 3.1.3.2 For subcase 3.1.3.2, where $p_3 < p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$, $t_1 + te_1 = \max\{p_1, t_1 + te_1, p_2\}$ and

$$(n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3) = \min\left\{(t_1 + te_1) - p_2, (n-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i + p_2 - p_3)\right\},$$

we define the following schedule (Schedule 3.1.3.2):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S_{1j}^m, C_{1j}^m] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_j on M_2 during time interval $[S_{2j}^m, C_{2j}^m] = [p_1 + t_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i), S_{2j}^m + p_2]$ for $j = 1, ..., n$.
- Schedule job J_j on M_3 during time interval $[S_{3j}^m, C_{3j}^m] = [\sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i), S_{3j}^m + p_3]$ for $j = 1, ..., n$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $[S_{1j}^r, C_{1j}^r] = [p_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i), S_{1j}^r + t_1]$ for $j = 1, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $[S_{2j}^r, C_{2j}^r] = [\sum_{i=1}^{2} p_i + t_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i), S_{2j}^r + t_2]$ for $j = 1, ..., n$.
- Move the robot empty from M_3 to M_1 during time interval $[\sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i), \sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + \sum_{i=1}^{2} te_i]$ for $j = 1, ..., n-1$.

Schedule 3.1.3.2 is illustrated in Figure 19 below for $n = 3$ with empty return moves shown in bold.

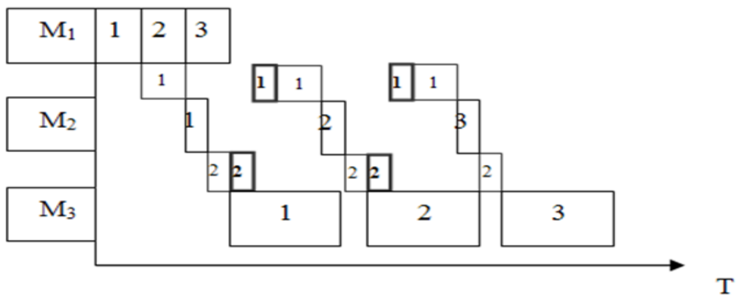

Figure 19: Schedule 3.1.3.2.

Lemma 19 *Schedule 3.1.3.2 is an optimal schedule for subcase 3.1.3.2.*

Proof. We start by showing that schedule 3.1.3.2 is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$; that

$$S^m_{2,j+1} = p_1 + t_1 + j(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) > C^m_{2j} = p_1 + t_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + p_2;$$

and that

$$\begin{aligned} S^m_{3,j+1} &= \sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + j(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) > \\ C^m_{3j} &= \sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + p_3, \end{aligned}$$

where the last inequality follows from the case condition that $p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i > p_3$, implies that there are no overlaps between processing of jobs in any machine. Moreover, the fact that $C^m_{1j} = jp_1 \leq S^r_{1j} = p_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i)$; that $C^r_{1j} = S^m_{2,j} = p_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + t_1$; that $C^m_{2j} = S^r_{2j} = p_1 + t_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + p_2$; and that
$C^r_{2j} = S^m_{3j} = \sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i)$, where the inequality follows from the fact that $p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i > p_3 \geq p_1$, implies that there is no overlap between processing and transferring operations of job J_j for $j = 1, \ldots n$. Thus, Condition 1 holds.

Since

$$C^r_{1j} = p_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + t_1 \leq S^r_{2j} = \sum_{i=1}^{2} p_i + t_1 + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i)$$

there is no overlap between the moves of J_j from M_1 to M_2 and J_j from M_2 to M_3. Moreover, after completing the move of J_j from M_2 to M_3, the robot returns to M_1 at time

$$\sum_{i=1}^{2} p_i + \sum_{i=1}^{2} t_i + (j-1)(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + \sum_{i=1}^{2} te_i = S^r_{1,j+1},$$

which implies that there is no overlap between the moves of J_j from M_2 to M_3 and J_{j+1} from M_1 to M_2. Therefore, there is no overlap between robot operations and Condition 2 holds.

The fact that $C^m_{1j} = jp_1 < S^r_{1j} = p_1 + (j-1)(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i)$ follows from the fact that $p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i > p_3 \geq p_1$, and implies that job J_j is ready to move from M_1 to M_2 at time S^r_{1j}. Moreover, the fact that $C^m_{2j} = S^r_{2j} = p_1 + t_1 + (j-1)(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i) + p_2$ implies that job J_j is ready to move from M_2 to M_3 at time S^r_{2j}. Thus, Condition 3 holds as well.

It is implied from the feasibility of schedule 3.1.3.2 that the completion time of job J_n on machine M_3 is at time $C^m_{3n} = \sum_{i=1}^2 t_i + \sum_{i=1}^2 p_i + (n-1)(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i) + p_3$. The fact that this time matches the lower bound in (eq. 87) when $t_1 + te_1 - p_2 > (n-1)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i + p_2 - p_3)$ implies that this schedule 3.1.3.2 is optimal for subcase 3.1.3.2. $\blacksquare$

9.3.2 The analysis of subcase 3.2

We further divide subcase 3.2, where $p_3 \geq p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i$, into two subcases. The first is subcase 3.2.1, is where $p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i \geq p_1$ and the second is subcase 3.2.2, the opposite case.

Optimal schedule for subcase 3.2.1 For subcase 3.2.1, where $p_3 \geq p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i$ and $p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i \geq p_1$, we define the following schedule (Schedule 3.2.1):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S^m_{1j}, C^m_{1j}] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_j on M_2 during time interval $[S^m_{2j}, C^m_{2j}] = [p_1 + t_1 + (j-1)\left(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right), p_1 + t_1 + (j-1)\left(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) + p_2]$ for $j = 1, ..., n$.
- Schedule job J_j on M_3 during time interval $[S^m_{3j}, C^m_{3j}] = [p_1 + t_1 + p_2 + t_2 + (j-1)p_3, p_1 + t_1 + p_2 + t_2 + jp_3]$ for $j = 1, ..., n$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $[S^r_{1j}, C^r_{1j}] = [p_1 + (j-1)\left(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right), p_1 + (j-1)\left(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) + t_1]$ for $j = 1, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $[S^r_{2j}, C^r_{2j}] = [p_1 + (j-1)\left(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) + t_1 + p_2, p_1 + (j-1)\left(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) + t_1 + p_2 + t_2]$ for $j = 1, ..., n$.
- Move the robot empty from M_3 to M_1 during time interval $[p_1 + (j-1)\left(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) + t_1 + p_2 + t_2, p_1 + (j-1)\left(p_2 + \sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i\right) + t_1 + p_2 + t_2 + \sum_{i=1}^2 te_i]$ for $j = 2, ..., n-1$.

Schedule 3.2.1 is illustrated in Figure 20 below for $n = 3$ with empty return moves shown in bold.

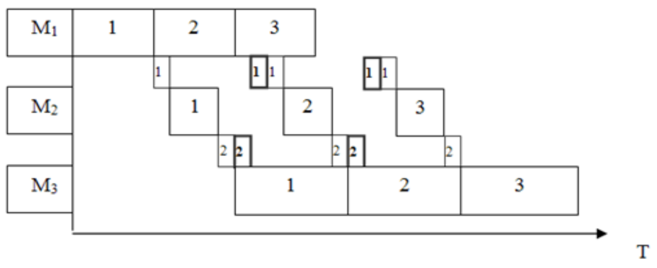

Figure 20: Schedule 3.2.1.

Lemma 20 *Schedule 3.2.1 is an optimal schedule for subcase 3.2.1.*

Proof. We start by showing that Schedule 3.2.1 is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$; that

$$S^m_{2,j+1} = p_1 + t_1 + j\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) \geq C^m_{2j} = p_1 + t_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + p_2;$$

and that $S^m_{3,j+1} = C^m_{3j} = p_1 + t_1 + p_2 + t_2 + jp_3$ implies that there is no overlap between the processing of different jobs on each machine. Moreover, the fact that $C^m_{1j} = jp_1 \leq S^r_{1j} = p_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$; that $C^r_{1j} = S^m_{2j} = p_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1$; that $C^m_{2j} = S^r_{2j} = p_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + p_2$; and that

$$C^r_{2j} = p_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + p_2 + t_2 \leq S^m_{3j} = p_1 + t_1 + p_2 + t_2 + (j-1)p_3,$$

where the first inequality follows from the fact that $p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i \geq p_1$, and the second inequality follows from the fact that $p_3 \geq p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$, implies that there is no overlap between processing and transferring operations of job J_j for $j = 1, ..., n$. Thus, Condition 1 holds.

The fact that $S^r_{2j} - C^r_{1j} = p_2 > 0$ implies that there is no overlap between the non-empty moves of the robot. Moreover, the fact that after downloading job J_j on M_3 the robot returns empty to M_1 at time

$$p_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) + t_1 + p_2 + t_2 + \sum_{i=1}^{2} te_i = p_1 + j\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right) = S^r_{1,j+1},$$

implies that there is no overlap between empty and non-empty moves. Thus, Condition 2 holds.

The fact that $C^m_{1j} = jp_1 \leq S^r_{1j} = p_1 + (j-1)\left(p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i\right)$ implies that job J_j is ready to move from M_1 to M_2 at time S^r_{1j}. Moreover, the fact that $C^m_{2j} = S^r_{2j}$ implies that job J_j is ready to move from M_2 to M_3 at time S^r_{2j}, and thus Condition 3 holds as well and Schedule 3.1.1 is feasible.

It is implied from the feasibility of Schedule 3.2.1 that the completion time of job n on M_3 is at time $C^m_{3n} = p_1 + t_1 + p_2 + t_2 + np_3$. The fact that this time matches the lower bound in (20) implies that this Schedule 3.2.1 is optimal for subcase 3.2.1. ■

Optimal schedule for subcase 3.2.2 For subcase 3.2.2, where $p_3 \geq p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i$ and $p_2 + \sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i < p_1$, we define the following schedule (Schedule 3.2.2):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S^m_{1j}, C^m_{1j}] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_j on M_2 during time interval $[S^m_{2j}, C^m_{2j}] = [jp_1 + t_1, jp_1 + t_1 + p_2]$ for $j = 1, ..., n$.
- Schedule job J_j on M_3 during time interval $[S^m_{3j}, C^m_{3j}] = [p_1 + t_1 + p_2 + t_2 + (j-1)p_3, p_1 + t_1 + p_2 + t_2 + jp_3]$ for $j = 1, ..., n$.

Robot Schedule:

- Move job J_j from M_1 to M_2 during time interval $[S^r_{1j}, C^r_{1j}] = [jp_1, jp_1 + t_1]$ for $j = 1, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $[S^r_{2j}, C^r_{2j}] = [jp_1 + t_1 + p_2, jp_1 + t_1 + p_2 + t_2]$ for $j = 1, ..., n$.
- Move the robot empty from M_3 to M_1 during time interval $[jp_1 + t_1 + p_2 + t_2, jp_1 + t_1 + p_2 + t_2 + \sum_{i=1}^{2} te_i]$ for $j = 2, ..., n-1$.

Schedule 3.2.2 is illustrated in Figure 21 below for $n = 3$ with empty return moves shown in bold.

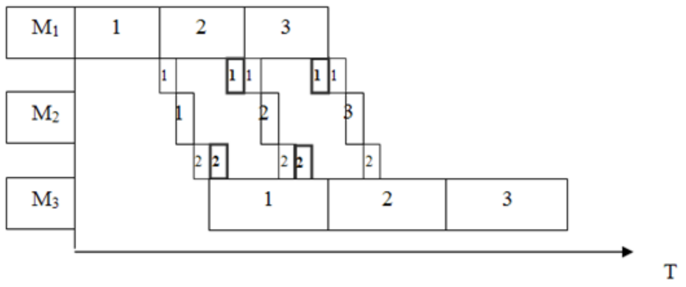

Figure 21: Schedule 3.2.2.

Lemma 21 *Schedule 3.2.2 is an optimal schedule for subcase 3.2.2.*

Proof. We start by showing that Schedule 3.2.2 is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$; that $S^m_{2,j+1} = (j+1)p_1 + t_1 > C^m_{2j} = jp_1 + t_1 + p_2$; and that $S^m_{3,j+1} = C^m_{3j} = p_1 + t_1 + p_2 + t_2 + jp_3$, where the inequality follows from that fact that $p_1 > p_2$, implies that there is no overlap between the processing of different jobs on each of the machines. Moreover, the fact that $C^m_{1j} = S^r_{1j} = jp_1$; that $C^r_{1j} = S^m_{2j} = jp_1 + t_1$; that $C^m_{2j} = S^r_{2j} = jp_1 + t_1 + p_2$; and that $C^r_{2j} = jp_1 + t_1 + p_2 + t_2 \leq S^m_{3j} = p_1 + t_1 + p_2 + t_2 + (j-1)p_3$, where the inequality follows from that fact that $p_3 \geq p_1$, implies that there is no overlap between processing and transferring operations of job J_j for $j = 1, ..., n$. Thus, Condition 1 holds.

The fact that $S^r_{2j} - C^r_{1j} = p_2 > 0$ implies that there is no overlap between the non-empty moves of the robot. Moreover, the fact that after downloading job J_j on M_3, the robot returns empty to M_1 at time $jp_1 + t_1 + p_2 + t_2 + \sum_{i=1}^{2} te_i < S^r_{1,j+1} = (j+1)p_1$ implies that there is no overlap between empty and non-empty moves. Thus, Condition 2 holds.

The fact that $C^m_{1j} = S^r_{1j} = jp_1$ implies that job J_j is ready to move from M_1 to M_2 at time S^r_{1j}. Moreover, the fact that $C^m_{2j} = S^r_{2j} = jp_1 + t_1 + p_2$, implies that job J_j is ready to move from M_2 to M_3 at time S^r_{2j}, and thus Condition 3 holds as well and Schedule 3.1.1 is feasible.

It is implied from the feasibility of schedule 3.2.2 that the completion time of job n on M_3 is at time $C^m_{3n} = p_1 + t_1 + p_2 + t_2 + np_3$. The fact that this time matches the lower bound in (20) implies that this schedule 3.2.2 is optimal for subcase 3.2.2. ■

9.4 The analysis of *Case 4*

We divide this case into two subcases. The first subcase 4.1 (9.4.1) is where $t_1 + te_1 \geq \max\{p_1, p_2\}$ and $t_2 + te_2 \geq \max\{p_2, p_3\}$, and the second subcase 4.2 (9.4.2) is the opposite case.

9.4.1 The analysis of subcase 4.1

For subcase 4.1, where $t_1 + te_1 \geq \max\{p_1, p_2\}$ and $t_2 + te_2 \geq \max\{p_2, p_3\}$, we define the following schedule (Schedule 4.1):

Machine Schedule:

- Schedule job J_j on M_1 during time interval $[S^m_{1j}, C^m_{1j}] = [(j-1)p_1, jp_1]$ for $j = 1, ..., n$.
- Schedule job J_1 on M_2 during time interval $[S^m_{21}, C^m_{21}] = [p_1 + t_1, S^m_{21} + p_2]$.
- Schedule job J_j on M_2 during time interval $[S^m_{2j}, C^m_{2j}] = [p_1 + 2t_1 + te_1 + (j-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i), S^m_{2j} + p_2]$ for $j = 2, ..., n$.
- Schedule job J_j on M_3 during time interval $[S^m_{3j}, C^m_{3j}] = [S^m_{2,j+1} + t_2, S^m_{3j} + p_3]$ for $j = 1, ..., n-1$.
- Schedule job J_n on M_3 during time interval $[S^m_{3n}, C^m_{3n}] = [S^m_{3,n-1} + te_2 + t_2, S^m_{3n} + p_3]$.

Robot Schedule:

- Move job J_1 from M_1 to M_2 during time interval $[S^r_{11}, C^r_{11}] = [p_1, S^r_{11} + t_1]$.
- Move job J_j from M_1 to M_2 during time interval $[S^r_{1j}, C^r_{1j}] = [p_1 + t_1 + te_1 + (j-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i), S^r_{1j} + t_1]$ for $j = 2, ..., n$.
- Move job J_j from M_2 to M_3 during time interval $[S^r_{2j}, C^r_{2j}] = [S^m_{2,j+1}, S^r_{2j} + t_2]$ for $j = 1, ..., n-1$.
- Move job J_n from M_2 to M_3 during time interval $[S^r_{2n}, C^r_{2n}] = [S^m_{3,n-1} + te_2, S^r_{2n} + t_2]$.
- Move the robot empty from M_2 to M_1 during time interval $[p_1 + t_1, p_1 + t_1 + te_1]$.
- Move the robot empty from M_3 to M_1 during time interval $[S^m_{3j}, S^m_{3j} + \sum_{i=1}^{2} te_i]$ for $j = 1, ..., n-2$.
- Move the robot empty from M_3 to M_2 during time interval $[S^m_{3,n-1}, S^m_{3,n-1} + te_2]$.

Schedule 4.1 is illustrated in Figure 22 below for $n = 3$ with empty return moves shown in bold.

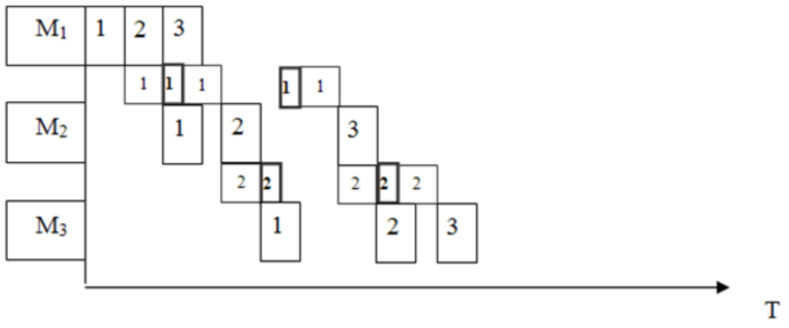

Figure 22: Schedule 4.1.

Lemma 22 *Schedule 4.1 is an optimal schedule for Case 4.1.*

Proof. We start by showing that schedule 4.1 is a feasible schedule. The fact that $S^m_{1,j+1} = C^m_{1j} = jp_1$; that $S^m_{22} = p_1 + 2t_1 + te_1 \geq C^m_{21} = p_1 + t_1 + p_2$ (since $t_1 + te_1 \geq p_2$); that

$$S^m_{2,j+1} = p_1+2t_1+te_1+(j-1)(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i) \geq C^m_{2j} = p_1+2t_1+te_1+(j-2)(\sum_{i=1}^{2} t_i+\sum_{i=1}^{2} te_i)+p_2$$

for $j = 3, ..., n-1$ (due to Case 4 condition that $\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i \geq p_2$); that

$$S^m_{3,j+1} = S^m_{2,j+2} + t_2 = p_1 + 2t_1 + te_1 + j(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + t_2 \geq C^m_{3j} =$$

$$S^m_{2,j+1} + t_2 + p_3 = p_1 + 2t_1 + te_1 + (j-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + t_2 + p_3$$

(due to Case 4 condition that $\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i \geq p_3$) for $j = 1, ..., n-2$; and that $S^m_{3n} = S^m_{3,n-1} + te_2 + t_2 \geq C^m_{3,n-1} = S^m_{3,n-1} + p_3$ (due to the fact that $t_2 + te_2 \geq \max\{p_2, p_3\}$), implies that there are no overlaps between processing of jobs in any machine. Moreover, the fact that $C^m_{11} = S^r_{11} = p_1$; that $C^m_{1j} = jp_1 < S^r_{1j} = p_1 + t_1 + te_1 + (j-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i)$ for $j = 2, ..., n$ (since $t_1 + te_1 \geq p_1$ and $\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i \geq p_1$); that $C^r_{11} = p_1 + t_1 = S^m_{21}$; that $C^r_{1j} = p_1 + 2t_1 + te_1 + (j-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) = S^m_{2j}$ for $j = 2, ..., n$; that $C^m_{21} =$

$p_1 + t_1 + p_2 \le S^r_{21} = p_1 + 2t_1 + te_1$ (since $t_1 + te_1 \ge p_2$); that

$$\begin{aligned} C^m_{2j} &= p_1 + 2t_1 + te_1 + (j-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + p_2 \le \\ S^r_{2j} &= S^m_{2,j+1} = p_1 + 2t_1 + te_1 + (j-1)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) \end{aligned}$$

for $j = 2, ..., n-1$ (since $\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i \ge p_2$); that

$$\begin{aligned} C^m_{2n} &= p_1 + 2t_1 + te_1 + (n-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + p_2 \le S^r_{2n} = S^m_{3,n-1} + te_2 = S^m_{2n} + t_2 + te_2 \\ &= p_1 + 2t_1 + te_1 + (n-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + t_2 + te_2 \end{aligned}$$

(since $t_2 + te_2 \ge p_2$); that $C^r_{2j} = S^m_{2,j+1} + t_2 = S^m_{3,j}$ for $j = 1, ..., n-1$; and that $C^r_{2n} = S^m_{3,n-1} + te_2 + t_2 = S^m_{3n}$, implies that there is no overlap between processing and transferring operations of job J_j for $j = 1, \dots n$. Thus, Condition 1 holds.

Once job J_1 is completed on M_1, the robot moves this job to M_2 during time interval $[p_1, p_1 + t_1]$ and returns empty to M_1 during time interval $[p_1 + t_1, p_1 + t_1 + te_1]$. Then, for $j = 2, ..., n$, the robot moves J_j from M_1 to M_2 during time interval $[p_1 + t_1 + te_1 + (j-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i), S^r_{1j} + t_1]$. This move is followed by the move of job J_{j-1} from M_2 to M_3 during time interval $[S^r_{2j}, C^r_{2j}] = [S^m_{2,j+1}, S^r_{2j} + t_2]$, which is followed either by an empty move to M_1 during time interval $[S^m_{3j}, S^m_{3j} + \sum_{i=1}^{2} te_i]$ if $j = 1, ..., n-2$, or by an empty move to M_2 during time interval $[S^m_{3,n-1}, S^m_{3,n-1} + te_2]$ if $j = n-1$. Lastly, the robot moves job J_n from M_2 to M_3 during time interval $[S^m_{3,n-1} + te_2, S^r_{2n} + t_2]$. The fact that

$$p_1 + t_1 + te_1 + (j-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + t_1 = S^m_{2,j+1} = p_1 + 2t_1 + te_1 + (j-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i)$$

for $j = 2, ..., n$; that $C^r_{2j} = S^m_{2,j+1} + t_2 = S^m_{3j}$; and that

$$\begin{aligned} S^m_{3,n-2} + \sum_{i=1}^{2} te_i &= S^m_{2,n-1} + t_2 + \sum_{i=1}^{2} te_i = p_1 + 2t_1 + te_1 + (n-3)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + t_2 + \sum_{i=1}^{2} te_i \le \\ S^m_{3,n-1} &= S^m_{2n} + t_2 = p_1 + 2t_1 + te_1 + (n-2)(\sum_{i=1}^{2} t_i + \sum_{i=1}^{2} te_i) + t_2; \end{aligned}$$

implies that there is no overlap between robot operations. Thus, Condition 2 holds.

The fact that $C^m_{11} = S^r_{11} = p_1$, and that $C^m_{1j} = jp_1 < S^r_{1j} = p_1 + t_1 + te_1 + (j - 2)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i)$ for $j = 2, ..., n$ (since $t_1 + te_1 \geq p_1$ and $\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i \geq p_1$), implies that job J_j is ready to move from M_1 to M_2 at time S^r_{1j}. Moreover, the fact that $C^m_{21} = p_1 + t_1 + p_2 \leq S^r_{21} = p_1 + 2t_1 + te_1$ (since $t_1 + te_1 \geq p_2$), and that

$$\begin{aligned} C^m_{2j} &= p_1 + 2t_1 + te_1 + (j-2)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i) + p_2 \leq S^r_{2j} = \\ S^m_{2,j+1} &= p_1 + 2t_1 + te_1 + (j-1)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i) \end{aligned}$$

for $j = 2, ..., n-1$ (since $\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i \geq p_2$), implies that job J_j is ready to move from M_2 to M_3 at time S^r_{2j}. Thus, Condition 3 holds as well.

It is implied from the feasibility of Schedule 4.1 that the completion time of job n on M_3 is at time

$$\begin{aligned} C^m_{3n} &= S^m_{3,n-1} + te_2 + t_2 + p_3 = S^m_{2,n} + t_2 + te_2 + t_2 + p_3 = \\ & p_1 + 2t_1 + te_1 + (n-2)(\sum_{i=1}^2 t_i + \sum_{i=1}^2 te_i) + t_2 + te_2 + t_2 + p_3 \\ &= p_1 + n\sum_{i=1}^2 t_i + (n-1)\sum_{i=1}^2 te_i + p_3. \end{aligned}$$

The fact that this time matches the lower bound in (4) implies that this Schedule 4.1 is optimal for subcase 4.1. ■

9.4.2 The analysis of subcase 4.2

We leave the analysis of this subcase to future research.

9.5 Examples and illustrations for the robot selection and scheduling problem

9.5.1 Single robot scheduling

An example of the robot move sequences of Hurink and Knust [38] for $n = 6$ and $m = 4$ is illustrated in Figure 23 , with empty return moves shown in bold.

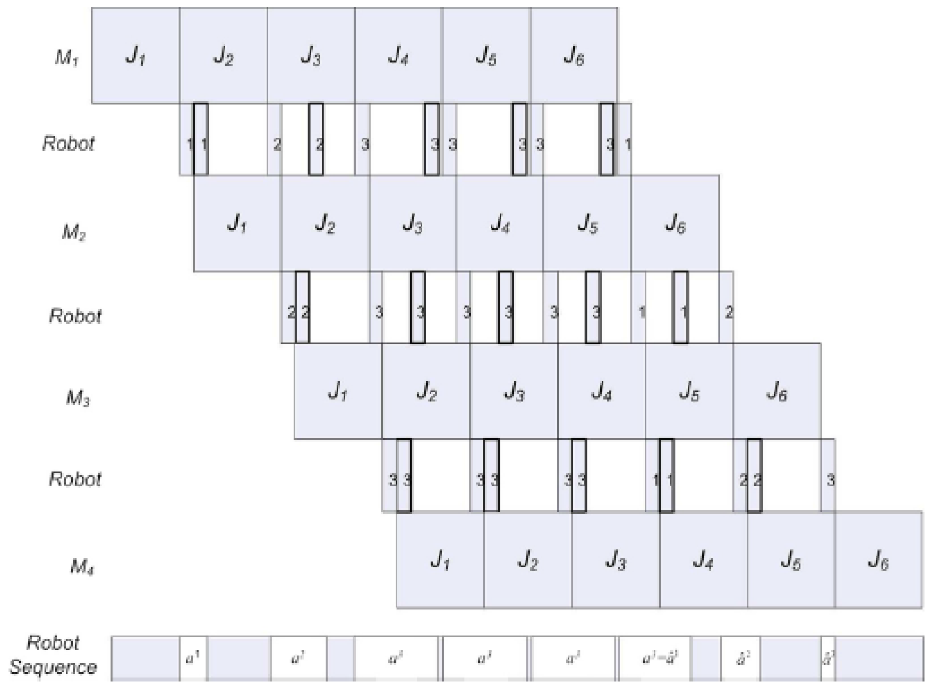

Figure 23: Illustration of a feasible robot schedule with m=4 and n=6.

The building-up phase in the illustrated example includes three sequences of moves (see first three robot sequences at the bottom of Fig. 23), where in the first sequence, a^1, the robot perform operation (J_1, M_1), and J_2 completes on M_1. In the 2nd sequence, a^2, the robot perform operations (J_2, M_1) and (J_1, M_2), and then J_3 completes on M_1. In the last sequence, a^3, within this phase the robot perform operations (J_3, M_1), (J_2, M_2) and (J_1, M_3), and then J_4 completes on M_1. The identity phase in the example includes two sequences of moves, where in the first sequence the robot perform operations (J_4, M_1), (J_3, M_2), (J_2, M_3), and then J_5 completes on M_1. In the second sequence the robot perform operations (J_5, M_1), (J_4, M_2) and (J_3, M_3), and then J_6 completes on M_1. The building-down phase in the illustrated example includes three sequences of moves, where in the first sequence, $\widehat{a}^1$, the robot perform operations $(J_6, M_1), (J_5, M_2)$ and (J_4, M_3), and then J_6 completes on M_2. In the second sequence, $\widehat{a}^2$, the robot perform operations (J_6, M_2) and (J_5, M_3), and then J_6 completes on M_3. The robot schedule ends with sequence $\widehat{a}^3$ where the robot perform operation (J_6, M_3).

9.5.2 An illustration of the polynomial reduction

Given a set of six machines ($m = 6$) and three different robot types ($Q = 3$). The transportation times and empty return times from each machine to the other and the cost of each robot type are given in Table 9 and $p = 10$.

q	t_{1q}	t_{2q}	t_{3q}	t_{4q}	t_{5q}	te_{1q}	te_{2q}	te_{3q}	te_{4q}	te_{5q}	δ_q
1	2	3	1	3	2	1	2	1	1	2	10
2	4	6	3	5	3	2	3	2	2	2	5
3	3	4	2	5	3	1	2	2	2	2	7

Table 9: Numerical Data

According to the polynomial reduction appearing within the proof of Theorem 2, construct a multigraph that includes $N = m = 6$ nodes (each of which represent a single machine in the production line). Furthermore, for any combination of a robot type q ($q = 1, 2, 3$) and machines M_u and M_v ($1 \leq u \leq 5$ and $u + 1 \leq v \leq 6$), if $\sum_{i=u}^{v-1}(t_{iq} + te_{iq}) \leq p$, include an arc $(u, v)_q$ in the set E which is directed from u to v and corresponds to the feasible assignment of a robot of type q to serve machines $M_u, ..., M_v$. The arc included in set E has length as given by eq. (54) and weight as given by eq. (55). To illustrate this procedure consider robot type $q = 1$ and machines $M_u = M_2$ and $M_v = M_4$. Since $\sum_{i=u}^{v-1}(t_{iq} + te_{iq}) = \sum_{i=2}^{3}(t_{iq} + te_{iq}) = (3 + 2) + (1 + 1) \leq p = 10$, include an arc $(2, 4)_1$ in set E. According to eqs. (54) and (55), the length of this arc is $\sum_{i=2}^{3} t_{i1} = 3 + 1 = 4$ and its cost is simply $\delta_1 = 10$. As an additional example, consider robot type $q = 2$ and machines $M_u = M_3$ and $M_v = M_5$. Since $\sum_{i=u}^{v-1}(t_{iq} + te_{iq}) = \sum_{i=3}^{4}(t_{i2} + te_{i2}) = (3 + 2) + (5 + 2) > p$, do not include arc $(3, 5)_2$ in set E. Fig. 24 illustrates the resulting multigraph.

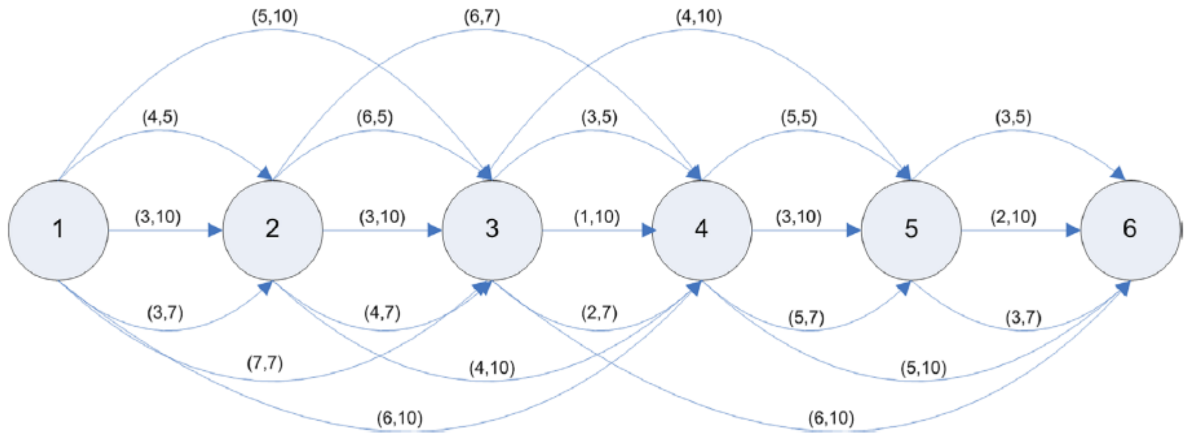

Figure 24: The Multigraph for the numerical example.

9.5.3 An illustration of the implementation of Algorithm 3

Following the numerical example in Appendix 9.5.2, in Steps $1-3$ of Algorithm 3 construct the simple directed acyclic graph, $G(\overline{V}, \overline{E})$ from the multigraph $G(V, E)$ that is illustrated in Fig. 24. In the construction, the weight of each arc is calculated by (56) and then replace a set of arcs connecting the same nodes with a single arc which corresponds to the minimum weight. For example, in Fig. 24 there are three arcs connecting nodes 1 and 2 with weights of $9, 13$ and 10. Thus, in $G(\overline{V}, \overline{E})$ a single arc with a length of 9 is included. The resulting directed acyclic graph is illustrated in Fig. 25.

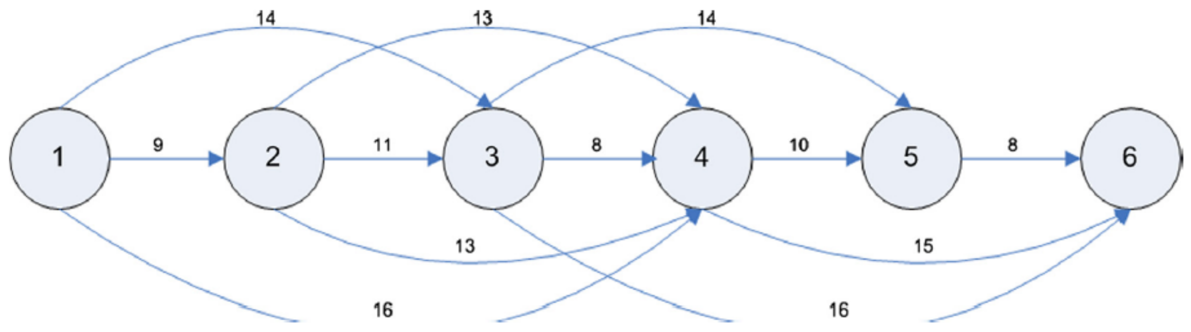

Figure 25: The simple graph for the example.

In Step 4, the recursion in (58) for $v = 1, \ldots, 6$ is applied with the initial condition $G_1 = 0$, and obtain $G_6 = 30$ as the length of the shortest path in $G(\overline{V}, \overline{E})$. The internal vertices of the shortest path are $l_1 = 1$, $l_2 = 3$ and $l_3 = 6$. Accordingly, a robot of type $q_{(1,3)}$ $= 3$ is assigned to serve machine set $\mathcal{M}_1 = \{M_1, M_2, M_3\}$ and robot of type $q_{(3,6)} = 1$ to serve machine set $\mathcal{M}_2 = \{M_3, M_4, M_5, M_6\}$.

9.5.4 An illustration of the implementation of Algorithm 7

To illustrate Algorithm 7, consider an instance similar to the numerical example in Appendix 9.5.2, with a single difference that is only includes robots of type 1 (i.e., robots of types 2 and 3 are not available here). In the initialization stage, set $l_1 = 1$ and $j = 1$. Then, in the first iteration of Steps 1 and 2, $a_1 = \arg \max\limits_{v=2,\ldots,m+1} \big\{\sum_{i=1}^{v-1}(t_i + te_i)\,\big|\sum_{i=1}^{v-1}(t_i + te_i) \leq 10\big\} = 4$, and thus set $l_2 = a_1 = 4$ and assign a robot to machine set $\{M_1, ..., M_4\}$. At the second iteration of Steps 1 and 2, $a_4 = \arg \max\limits_{v=5,\ldots,m+1} \big\{\sum_{i=4}^{v-1}(t_i + te_i)\,\big|\sum_{i=4}^{v-1}(t_i + te_i) \leq 10\big\} = 6$, and thus set $l_2 = a_1 = 6$ and assign a robot to machine set $\{M_4, ..., M_6\}$. Since $l_2 = m$, all machines are covered and the algorithm ends.

9.6 *RA* and *RR RL* collaboration pseudo-code

RA RL collaboration pseudo-code

```
    Input: $n$, $m$, IM or NIM, $[t_i, \Delta]$, $[te_i, \Delta]$, $\acute{E}$, $X$, $Temp$ (constant or dynamic).
Step 1. Initialize $Q(s,a) = 0$ for a learning session (i.e. $|\acute{E}|$ episodes learning)
Step 2. Repeat (for each learning episode $\acute{E}_i$ $i = 1, ..., \acute{E}$):
                    Initialize state $s_t$ as starting state $s_t = [2, 1, ..., 1]$ and $t^1$
                    Repeat (for each step of episode):
                        Calculate $\varepsilon$ (eq. 69)
                        if Sample from $U\tilde{}[0,1] < \varepsilon$
                            Perform Random action
                        else
                            Calculate $Q_t(s,a)$ (eq. 70) for feasible actions
                            Perform Greedy action, i.e. $\max Q_t(s,a)$
                        End if
                        Update $a_t$, $Q_t(s,a)$, $s_{t+1}$ and System transition time $t^{t+1}$
                                                                  using $\tau$ (eq. 67)
                    Until a stopping condition (i.e. reached goal state - $S_{\check{T}}$)
                    if number episode learning $i > X$
                      Calculate $C_{ave}$ (eq. 71)
                        If $C_{ave} > \hat{C}_{\max}$ and Adviser is Not Discarded
                            Collaborate Adviser (i.e., Perform RL algorithm
                                      using Softmax Action Selection)
                            if $C_{\max}$(adviser) $< \hat{C}_{\max}$
                              Perform Adviser Advice (Schedule)
                                and Update $Q_t(s,a)$
                            End if
                          Update $Ha$ (eq. 72)
                          Evaluate Adviser Expertise and
                          Update Discard or Keep Adviser?
                        else
                            if $C_{ave} < \hat{C}_{\max}$ then Update $\hat{C}_{\max} = C_{ave}$
                        End if
                    End if
                    At end of $E_1$ Update $\hat{C}_{\max} = C_{\max}(\acute{E}_1)$
              Until a stopping condition (desired number of learning episodes $|\acute{E}|$)
```

RR RL collaboration pseudo-code

```
Input: $n, m$, IM or NIM, IR or NIR (IRT or NIRT), $[p_i, \Delta]$, $[t_i, \Delta]$, $\acute{E}$, $X$, $B$.
Step 1. Initialize $Q_1(s,a) = Q_2(s,a) = 0$ for a learning session (i.e. $|\acute{E}|$ episodes learning)
Step 2. Repeat (for each learning episode $\acute{E}_i$ $i = 1, ..., \acute{E}$):
            Initialize state $s_t$ as starting state $s_t = [2, 1, ..., 1]$
                    and $t^1 = \min(\tau_1, \tau_2)$
            Repeat (for each step of episode):
                For $R_k$ $(k = 1, 2)$ with $\min(\tau_1, \tau_2)$
                Calculate $\varepsilon$ (eq. 69)
                if Sample from $U\tilde{}[0,1] < \varepsilon$
                    Perform Random action
                else
                    Calculate $Q_t(s,a)$ (eq. 70) for feasible actions
                        Perform Greedy action, i.e. $\max Q_t(s,a)$
                End if
                Update $Q_{k,t}(s,a)$ (eq. 70), $a_{k,t+1}$, $s_{t+1}$, $\tau_k = \tau + t_i$ ( $\tau$ eq. 67)
                        and System transition time $t^{t+1} = \min(\tau_1, \tau_2)$
            Until a stopping condition (i.e. reached goal state - $S_{\check{T}}$)
            if number episode learning $i > X$
                Calculate $C_{ave}$ (eq.77)
                If $C_{ave} > \hat{C}_{\max}$
                    For Full / Push / Pull Activate Collaboration mode
                    For None Stay in Autonomous mode
                else
                    Activate Autonomous mode
                    Update $\hat{C}_{\max} = C_{ave}$
                End if
            End if
            if learning episode / collaboration resulted $C_{\max}(\acute{E}_1) < \hat{C}_{\max}$
                For all $Q_t(s,a)$ visited during the learning episode:
                    Update $Q_{1,t}(s,a) \leftarrow Q_{1,t}(s,a) + B$
                    Update $Q_{2,t}(s,a) \leftarrow Q_{2,t}(s,a) + B$
            End if
            At end of $\acute{E}_1$Update $\hat{C}_{\max} = C_{\max}(\acute{E}_1)$.
    Until a stopping condition (desired number of learning episodes $|\acute{E}|$)
```