2014

Policy based reinforcement learning approach Of Jobshop scheduling with high level deadlock detection

Mengmeng Chen
Iowa State University

A policy based reinforcement learning approach

for

jobshop scheduling with high level deadlock detection

by

Mengmeng Chen

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Industrial Engineering

Program of Study Committee:

Sigurdur Olafsson Major Professor

Hu Guiping

Wu Huaiqing

Iowa State University

Ames, Iowa

2013

Table of Contents

List of Figures

List of Tables

ABSTRACT

We present a policy based reinforcement learning scheduling algorithm with high level deadlock detection for job-shop discrete manufacturing systems without buffer being equipped. Deadlock is a highly undesirable phenomenon resulting from resource sharing and competition. Hence, we first propose detection algorithms for second and third level deadlocks. Subsequently, based on these high level deadlock detection algorithms, a new policy based reinforcement learning scheduling algorithm is developed in the context of buffer-less job-shop systems. Applying our reinforcement learning approach into scheduling algorithm to a set of 40 widely-used buffer-less job shop benchmark, satisfactory makespan can be obtained, which, to our knowledge, have never been published before. It is safe to conclude that our policy based reinforcement learning scheduling algorithm can be applied to other discrete event systems (e.g., computer operation systems, communication systems, and traffic systems).

***Keywords* –** High Level Deadlock Detection, Buffer-Less Job Shop scheduling, Policy based Reinforcement Learning

CHAPTER I OVERVIEW

1.1 Introduction to Deadlock in Job shop scheduling

Recently, there has been a significant interest in designing and developing flexible manufacturing systems (FMSs) among manufacturers so as to maintain the competitiveness. FMSs provides the manufacturers with great flexibility in meeting the demand, especially for low-volume and high variety products. However, since FMSs exhibits a high degree of resource sharing, manufacturing shops are subject to system deadlock, which may cause unnecessary costs due to long downtime and low resource utility.

Deadlock (DL) is a highly undesirable phenomenon in most discrete event systems (DES), which requires special managing and controlling strategies [3]. Essentially, DL results from resource sharing [6]. When various customers passing through a buffer-less service system and competing for limited resources, the system is prone to DL, an insidious halting condition in which there is a set of customers awaiting the allocation of resources held by other customers in the same set. The phenomenon is usually named as “circular waiting” or “hold and wait”. Since no buffer is equipped, a set of customers and servers is restricted in a fixed loop, and cannot be released via any feasible action within the loop.

With this knowledge, it is highly desirable, if not urgent, to solve the DL problems in FMSs while maintaining the viability of the system performances such as makespan. With this objective in mind, a new DL-free scheduling algorithm is proposed in this paper, consisting of a high-level DL detection method as well as a reinforcement learning approach. Specifically, in the context of job shop systems, the high-level DL detection method helps to reduce the searching time for a DL-free schedule. The experimental results show that, for most of the large-size job shop systems, a DL-free schedule can be found within seconds, which enables the real-time control of manufacturing shop when DL concern is being incorporated. At the same time, the reinforcement learning approach facilitates the improvement of the performances of the DL-free schedules, such as the makespan being considered in this paper. We notice that reinforcement learning approach has been studied by previous researchers in computer science in general and in manufacturing scheduling in particular. To our knowledge, Zhang and Dietterich [44] first applied the reinforcement learning approach in job shop scheduling problem (JSSP). Reinforcement learning is a machine learning approach to find a policy π which can maximize expected future return, which calculated based on reward function γ. Policy π determines which action will be choose by RL agent, and is usually state dependent [45]. A great deal of research has been invested into the development of JSSP solution methods both in the operations research and artificial intelligence communities. However, these previous works are all based on the assumption of unlimited buffer, or in other words, DL is not being considered during the scheduling process. On the other hand, limit-buffer JSSP or buffer-less JSSP has not studied in the extant literature whereas there is indeed a request from industry regarding the DL-free scheduling in job shop systems

Nowadays, several new application areas corresponding to some special FMSs are buffer-less systems in real production cases: (i) the shop floor of gigantic workpiece such as 'head of air-plane' or 'shaft of turbine dynamo' has no spare space for intermediate work in process (WIP), (ii) to keep high production efficiency in the IC chips photolithographic system, any block during chip transferring within different equipment chambers (of cluster tool) is unacceptable under the high chip delivering and handling rate. Hence, in these FMSs, resource competitions and DLs may occur frequently without well designed job scheduler and manufacturing controller.

Under this circumstance, as various constraint programming methods, combinatorial algorithms, and evolutionary algorithms have been widely utilized in schedule searching, ordinary FMSs (with sufficient buffer capacity and so without DL happening) scheduling has become a relatively matured technique as stated in extant literature [13, 17, 22, 25]. Nevertheless, for buffer-less FMSs, the scheduling problem is still not well resolved, even for small-scale FMSs. For example, to the best of our knowledge, for a set of widely-used benchmark FMSs (which will be introduced later), no DL-free schedule is published in the previous literature.

Basically, job shop FMS has the following two features: (i) the system resources are limited and not exemplified, and (ii) each customer is handled in accordance with its scheduled route (which may result in complex route crosscut). To satisfy these two features, well designed schedule is expected to handle the competition among customers.

As numerous research works have shown, schedule for a job shop FMS must be found based on the route constraints of customers and the system performing objectives (e.g., makespan, costs, and due-dates) required by user. The main task of FMSs controller [40] is to execute the obtained production schedule that can realize those requirements and achieve more benefits for FMSs user.

1.2 Research Objective

However, without high level DL detection algorithm, DL-free scheduling for buffer-less job shop FMSs has not been systematically solved. Hence, the objectives of this paper are

(1) To design and develop high level (second level and third level) DL detection algorithms

(2) To apply reinforcement learning approach in DL-free scheduling so as to improve the solution performances such as makepsan.

(3) To propose DL-free scheduling algorithm for job shop systems, consisting of the high-level DL detection methods, as well as the reinforcement learning approach.

1.3 Thesis Organization

The rest of the thesis is organized as follows: Chapter II presents the previous works related to this paper. In Chapter III, we introduce the first, second and third level DL patterns as well as the corresponding detection methods. In Chapter IV, we propose a policy based reinforcement learning deadlock free scheduling algorithm for buffer-less job shop FMSs. Chapter V provides the results of experiments on benchmark problems as well as implications drawn from these results. Finally, conclusion and future works are provided in Chapter VI.

CHAPTER II LITERATURE REVIEW

In this chapter, we review some literatures on Deadlock detection problems and reinforcement problems.

2.1 Job shop Scheduling Deadlock Detection

Single unit resource allocation system (SU-RAS) [23] is widely considered while modeling FMSs. Several researchers utilized a pioneer method of empty siphon detection for first level DL (relative to higher level DL in this paper) checking [7, 21]. Similarly, under the modeling frameworks of ROPN (Petri Net) [31, 32] and transition graph (Digraph) [8, 9, 10], a DL-free unsafe state can also be detected via using one-step ahead strategy. However, these studies are applicable to some specific small-scale FMSs only [1, 16, 37, 38]. A general DL-free scheduling algorithm for FMSs based on high level DL detection has never been proposed systematically.

Furthermore, automata model is also utilized to analyze DL in the context of computer operations system [39]. However, modeling the distributed features of job shop FMSs using automata cells and the corresponding control supervisor is usually intractable. Hence, the modeling methodology proposed in [39] is only applicable to small-scale FMSs with simple process routes. Fanti and Zhou [11] provided a complete survey in this area.

Given the modeling frameworks in the extant literature, three strategies to handle DL and the corresponding research works are illustrated as follows:

(1) Deadlock Prevention, which restrains the resource allocation for customers so that DL can be prevented. For Petri nets model, synthesis methods (e.g., mixed integer programming [5] and region theory [14]) are used to find siphon or elementary siphon [7, 21, 36] so as to prevent DL. In automata model, supervisory controllers [24] are introduced to keep systems in a DL-

free status. However, both synthesis methods and supervisory controllers require enormous computation consumption, which makes these methods intractable when applying to FMSs with considerable scale and complicated job shop route constraints.

(2) Deadlock Avoidance, which restrains the way resources are allocated based on the current state of the system and the next request of resource for each customer. DL avoidance algorithms are well developed in Petri Net model [2, 23, 29, 30, 32], Automata model [39], and Digraph model [4, 12, 18, 34] to handle FMSs scheduling problems. However, current DL avoidance algorithms can not avoid DL-free unsafe state (i.e., high level DL in this paper). Furthermore, several papers [10, 26, 31] try reserving one resource to avoid DL-free unsafe state. Nevertheless, by now, no DL avoidance algorithm is available to deal with high level DL. Hence, current DL avoidance algorithm is only applicable to FMSs with simple resources sharing structure (without high level DL happening).

(3) Deadlock Detection and Recovery, which concentrates on the expedient resolution after DL has been detected. Wysk [4, 33] introduced a string multiplication algorithm to detect DL for buffer-less job shop systems. Generally, an additional buffer is required to re-allocate the resources and resolve the detected DL. However, as introduced in Section 1, for some specific systems (e.g., air plane head and turbine dynamo shaft), there is no space for additional buffer, and resolving DL may bring great inconvenience.

It is well known that DL avoidance and detection in the context of finite-state system is a polynomial problem. Actually, DL-free scheduling is a kind of DL prevention, which requires all the customers to reach their end states without DL occurring. As the safety (or Reachability) problem in FMSs is a non-polynomial problem [15, 27], finding a DL-free schedule for a FMS is extremely difficult. Hence, for medium- or large-scale buffer-less job shop FMSs, such a scheduling problem has not been systematically solved in the previous research.

Additionally, inspired by DL-free scheduling algorithms utilized in computer operation system, a Banker's method [19] is used to allocate resources within different tasks without meeting DL. Banker's method may find DL-free schedule for large-scale buffer-less job shop FMSs. However, Banker's method is conservative and may reduce resources utility and processing efficiency in FMSs, and the obtained DL-free schedule tends to be overly restricted [19].

Despite of the methods mentioned above, Ramaswamy and Joshi [43] formulated the DL-free scheduling as an integer programming problem and were able to find the optimal total flow time of the DL-free schedule for small-size job shop systems. Specifically, based on the typical job shop scheduling integer programming framework, they added a constraint ensuring that a job leaves a resource only when it has found space on the next resource. It can be verified that their integer programming model provides an optimal DL-free schedule for job shop system with no buffer in terms of total/average flow time.

However, utilizing this integer programming method, only small size job shop scheduling problem can be solved in acceptable time. The integer programming formulation is shown below:

X_{iK} : *Completion time of last operation of job* i *(term* K*)*

X_{ikj} : *Completion time of job* i *operation* k *on machine* j;

T_{ikj} : *processing time of job* i *operation* k *on machine* j;

$Y_{prqsj} = \{0,1\}$: 1 *if job* p *operation* r *follows job* q *operation* s *on machine* j;

0 *if otherwise.*

H : *big positive number*

E : *small positive number*

I : *Set of all jobs* $\{1,2,3...N\}$

J : *Set of all machines* $\{1,2,3...M\}$

Formulation :

$$Min\ \ Z = \sum_{i=1}^{N} X_{iK}$$

$$X_{ikj} - T_{ikj} \geq X_{i(k-1)l} \quad [\forall i \in I; (X_{i(k-1)l} \textit{ is the completion time of job } i \textit{ operation } k-1 \textit{ on machine } l, l \in J)] \quad (1)$$

$$X_{prj} - X_{qsj} + H(1 - Y_{prqsj}) \geq T_{prj} \qquad [\forall j \in J; \forall (p,q) \in I] \qquad (2)$$

$$X_{qsj} - X_{prj} + HY_{prqsj} \geq T_{qsj} \qquad [\forall j \in J; \forall (p,q) \in I] \qquad (3)$$

$$X_{p(r-1)l} - X_{qsj} + H(1 - Y_{prqsj}) \geq E \quad [\forall j \in J; \forall (p,q) \in I, l \in J] \qquad (4)$$

$$X_{q(s-1)w} - X_{prj} + HY_{prqsj} \geq E \qquad [\forall j \in J; \forall (p,q) \in I, w \in J] \qquad (5)$$

In the Experiment section of this paper, we will show that, for job shop systems with 15 jobs or more and 5 resources or more, a DL-free schedule cannot be obtained in 48 hours on 64-bit operating system with 3.00GHZ CPU. Meanwhile, incorporating our high level DL detection method into the DL-free scheduling algorithm, even large-size (e.g., 30 jobs and 10 resources) job shop scheduling problem can be solved within seconds, and the obtained total flow time is reasonably acceptable.

2.2 Policy Based Reinforcement Learning

Reinforcement learning is a real time machine learning derived from the conventional optimal control technique known as dynamic programming. The environment is formulated as a discrete-time, finite state, markov decision process. The goal of this process is to solve the learning problem by finding the best policy of action, regardless of the deeper structure of the experience gathered during interacting with the environment. Zhang and Dietterich [44] were the first to apply a reinforcement learning approach for a special scheduling problem. They developed a repair-based scheduler that is trained using the temporal difference reinforcement learning algorithm starting with a critical-path schedule, and incrementally repairs the violations of the constraints. Mahadevan et al. [46] presented an average-reward reinforcement learning algorithm for the optimization of transfer lines in manufacturing systems which incorporates a simplifying specialization of a scheduling problem. They show that the adaptive resources are able to effectively learn when they have to request maintenance, and that introducing a hierarchical decomposition of the learning task is beneficial for obtaining superior results. Another repair-based approach based on an intelligent computing algorithm is suggested by Zeng and Sycara [47] who utilized case-based reasoning and a simplified reinforcement learning algorithm to achieve adaptation in changing optimization criteria.

However, the reinforcement learning approach utilized in the job-shop scheduling algorithm proposed in this paper,formulate the problem as a sequential decision problem and a Markov decision process, which is different from the previous works mentioned above. Similar approach has been utilized by Schneider et al. [48] in manufacturing scheduling problems. However, they assume a single learning agent that fully observes the state.

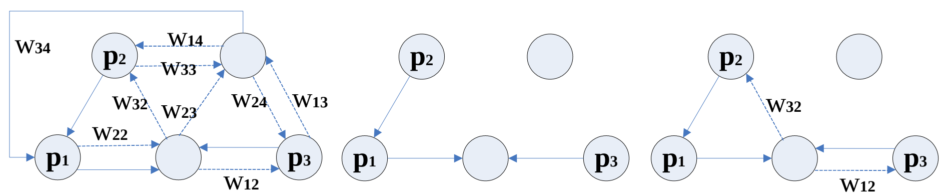
CHAPTER III. HIGH LEVEL DEADLOCK DETECTION

For job shop DMS, each product in the system must occupy one resource and each resource can process only one product at a time. Generally, the concerned DMS is running under conditions of: (i) mutation exclusion, (ii) no preemption and (iii) hold while wait. Since each product has to follow its own routine path in DMS and M products keep M independent routes, deadlocks may occur very often if we expect that resources are highly utilized. In this chapter we will introduce 3 different deadlock detection methods.

3.1 Deadlock Detection based on Digraph Method

Comparing with different DMS models, we believe digraph model, especially the modified transition digraph (MTG) introduced in this paper, is the most compact one to illustrate coupling relationships of products' routines. Our detection algorithm developed later in this section is based on the up-to-date state of DMS and its modified transition graph correspondingly. Generally, the immediately detected DL is defined as 1st level DL. As the implicit unsafe state may appear several steps later, we define them as 2nd level, 3rd level and higher level respectively.

It is well known that a static routine digraph (RG) that contains all processing paths in sequential is difficult to be used for model analysis since there are too many connection edges (for a job-shop system with M product and N resources, there is at most $M \times N$ edges) existing to represent all the handling steps and most of them are not in use at the moment. While transition digraph (TG) defined in [8,18] only gives the instant resource occupation and indicates the possible moving tendencies of each product under current state, which may be not enough for the analysis of potential deadlock. Therefore, in this paper, we define a specially "modified transition graph" which extends labeling some related edges to several future steps on ordinary TG, so that one may take more efficient analysis since those steps that may cause system DL in future are under consideration. A simple DMS (of 3 products and each has 4 routine steps within 5 resources) is given to show relation among RG, TG and MTG in Fig. 3.1



(a) Routine Graph (b) Transition Graph (c) Modified Transition Graph

Figure 1. RG, TG and MTG Model of a Simple DMS

As shown in Fig. 3.1(a), RG is too complicated and many edges within it are actually no use for detecting DL and potential DL. On the opposite, according to the TG model in Fig. 3.1(b), a potential deadlock will be omitted. MTG model in Fig. 3.1(c) removes futile future transitions and keeps some of them for related key products to help detecting so-called deadlock-free unsafe state that may possibly cause deadlock several steps later.

By using idea of MTG, DL-detection rule can be developed via the following steps:

(i) Choosing an unblocked transition (including a product entering into the system or a product which is already in the DMS moving from one resource to another) and virtually firing this transition, then a virtual system state is generated and the MTG correspondingly.

(ii) Analyzing the virtual MTG via several developed DL detection algorithm (introduced later this section) to detect whether there is a DL or potential DL (with different level) existing,

(iii) Recover the system to the state before virtually firing with the result of deadlock detection.

We must mention that the result of DL detection is still problematic, since the DL detection algorithms used in (ii) can only capture 1st, 2nd and 3rd level DL in the system. The transitions pass DL detection algorithms do not ensure the DMS is DL-free in future since potential DLs higher than 3rd level may be caused by the supposed feasible transition and still keep "DL-free unsafe states" of DMS in execution.

Nevertheless, by using 2nd and 3rd DL detections, the feasible state space of DMS can be greatly reduced during searching. From all the "deadlock-free" transitions, we may spend less backtracking computation efforts and have higher probability in finding an off-line DL-free schedule in this reduced state space. For most small and middle scale system (in the range of benchmark problems), the results obtained in our paper are quick, effective and applicable. We will explain them in detail in section 6.

For the convenience of analysis, we define some symbols as follows:

$HR(p_j)$	the resource which product $\mathbf{p_j}$ is holding
$FRR(p_j)$	the resource which product $\mathbf{p_j}$ requests next step
$SRR(p_j)$	the resource which product $\mathbf{p_j}$ requests two steps later
$TRR(p_j)$	the resource which product $\mathbf{p_j}$ requests three steps later

3.1.1 First Level Deadlock

The definition of first level deadlock is given as follow:

Definition 1.1: P_D is a set of products, if for any product $p_i \in P_D$, $FRR(p_i)$ is hold by another product $p_{i'} \in P_D$, and none of the proper subsets of P_D possesses the same property, then we define P_D as **first level deadlock**.

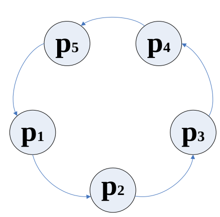


Figure 2. 'circular waiting' structure of 1st level DL

Actually, first level deadlock is a circular waiting cycle of products, and Fig. 3.2 gives a simple five-product first level deadlock example. For 1st level deadlock detection, we need to decide if such a loop structure existing at current state. Since first level deadlock detection algorithm may find in many references [10, 26, 33], we just drop it from our paper.

Definition 1.2: For a set of products P_C, if there exists an order for all the products in P_C: p_1, $p_2 \ldots p_{n'}$ (n' is the number of products in P_C) that $HR(p_{i+1}) = FRR(p_i)$ for any $i \in [1, n'-1]$ and $p_{n'}$ is not in its last processing step, then we define P_C as **potential deadlock chain (PDC)**. We define p_1 as the **start product of PDC**, and use p_S to denote it; define $p_{n'}$ as the **end product of PDC** which is denoted by p_E.

Following are some explanations about PDC which are useful for high level DL discussion later:

(1) A PDC has its unique start and end product, and they can be the same product. Fig. 3.3(a) shows the simplest instance of PDC, which has only one product.

(2) For a PDC, $FRR(p_E)$ can be either idle or occupied. Based on (1) and (2), we can find all the PDCs in products string shown in Fig. 3.3(b). Let's taking p_i ($i = 1 \ldots n$) as a start product, every p_i ($i \leq j \leq n$) can be the end product correspondingly. Thus, the total number of different PDCs in Fig. 3.3(b) is $\frac{1}{2}n(n+1)$.

(3) Start product p_S and end product p_E are the most important product in a PDC, while other products in PDC are usually ignored when analyzing deadlock patterns. So Fig. 3.3(c) is used to denote a PDC in general and will be used to represent PDC in the following figures.

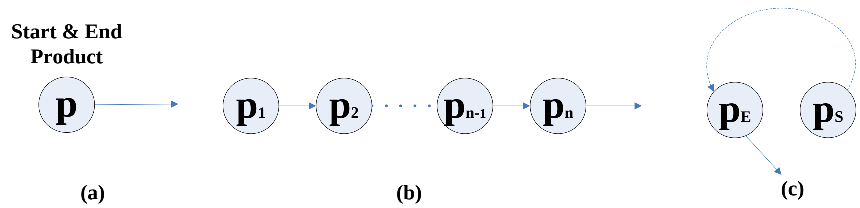


Figure 3. Instances of Potential Deadlock Chain (PDC)

3.1.2 Second Level Deadlock

Definition 1.3: P_{SD} is a set of products without 1st level DL, if after moving any movable product $p \in P_{SD}$, there will be another set of products $P_D \subseteq P_{SD}$, P_D is first level deadlock and none of the proper subsets of P_{SD} possesses the same property, then we define P_{SD} as **second level deadlock.**

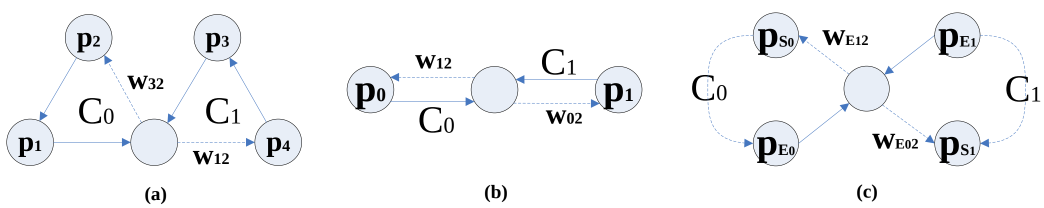


Figure 4. Instances of 2nd Level DL

Based on a reduced example from [8] shown in Fig. 3.4(a), one may understand 2nd level DL clearly. In Fig 3.4(a), there are 6 PDCs: $\{p_1\}$, $\{p_2\}$, $\{p_3\}$, $\{p_4\}$, $\{p_2, p_1\}$, $\{p_4, p_3\}$. We consider PDC $\{p_2, p_1\}$ and PDC $\{p_4, p_3\}$, the end of both PDCs request a common resource next step ($FRR(p_1) = FRR(p_3)$) and request the start of the other PDC two steps later ($SRR(p_1) = HR(p_4)$ and $SRR(p_3) = HR(p_2)$), which makes the two PDCs C_0 and C_1 coupling in the common idle resource. If replacing C_0 and C_1 with the simplest one-product PDC, we will obtain the simplest 2nd level deadlock as shown in Fig. 3.4 (b). Actually, the 2nd level DL instances in Fig. 3.4(a) and (b) both belong to the pattern shown in Fig. 3.4(c), which represents the general pattern of two-PDC 2nd level DL.

Intuitively, we can have more PDCs coupled in the common idle resources as shown in Fig. 3.5, which is the general pattern of 2nd level deadlock. In Fig. 3.5, we use C_K (K = 0, 1…n) to denote a PDC, p_{SK} is the start product of C_K, while p_{EK} is the end product.

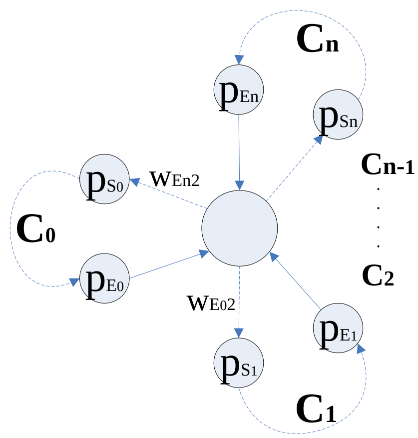


Figure 5. General Pattern of 2nd Level Deadlock

Proposition 1.4: Any second level deadlock is a structure of n ($n \geq 2$) PDCs coupling in a common idle resource, and for a structure of n ($n \geq 2$) PDCs coupling in a common idle resource, it is a second level deadlock.

Here we define "couple" in detail. Assume these n PDCs are: C_1, C_2, …, C_n. Then "couple" means $FRR(p_{E1}) = FRR(p_{E2}) = ... = FRR(p_{En}) = R_I$ (here R_I is the common idle resource) and $HR(p_{S(i+1)}) = SRR(p_{Ei})$ for $i \in [1, n\text{-}1]$ ($HR(p_{S1}) = SRR(p_{En})$). The proof of Proposition 1 is given as follow:

Necessity:

Obviously, if n PDCs couple in one common idle resource, then only the end product of every PDC can be moved. According to the definition of "couple", moving p_{Ei} (i = 1, ... n) will definitely cause deadlock immediately, and if any product in these n PDCs is removed, the left products can not form a second level deadlock. Therefore, a structure of n ($n \geq 2$) PDCs coupling in a common idle resource forms a second level deadlock and the necessity is proven.

Adequacy:

According to definition 3, if a set of products P_{SD} is 2nd level deadlock, then a subset of P_{SD} will be 1st level deadlock after moving any movable product belonging to P_{SD}. Assume p_T is a movable product in P_{SD} and P_D, a sub subset of P_{SD}, will become first level DL after moving p_T, then we can easily deduce that $p_T \in P_D$ and $P_{D*} = P_D - \{p_T\}$ is a PDC. Here we define P_{D*} as C_1 and define FRR(p_{E1}) as common idle resource R_I (as the idle resource in the center of Fig. 3.5).

Since moving any product belonging to P_{SD} will cause a first level DL, if we move p_{E1} to R_I, it will also cause a first level DL, which means that SRR(p_{E1}) must be occupied by the start product of another PDC which we name as C_2 (SRR(p_{E1}) = HR(p_{S2})), and FRR(p_{E2}) = FRR(p_{E1}) = R_I. Recursively, we will have PDCs $C_3, C_4, \ldots, C_n$ that HR($p_{S(i+1)}$) = SRR(p_{Ei}) for $i \in [1, n-1]$ and FRR(p_{E1}) = FRR(p_{E2}) = ... = FRR(p_{En}) = R_I. Because products number and resource number in the system are limited, there are only two possibilities for SRR(p_{En}):

1) SRR(p_{En}) = HR(p_{Si}) ($i \in [2, n-1]$), then there exists a set of products $P_{SD'} = C_i \cup \ldots \cup C_n$ that $P_{SD'} \subset P_{SD}$ and moving any movable product belonging to $P_{SD'}$ will cause first level deadlock, which violates the definition of 2nd level deadlock. So if P_{SD} is second level deadlock, SRR(p_{En}) = HR(p_{Si}) ($i \in [2, n-1]$) must be invalid.
2) SRR(p_{En}) = HR(p_{S1}), then n PDCs couple in R_I and the adequacy is proven.

Therefore, proposition 1 was proven and the structure shown in Fig. 3.5 is the general pattern of second level deadlock.

In order to detect whether there is second level deadlock in the system, one by one, we assume every product processing in the system to be the end product of a PDC, and check if there exists a second level deadlock from that PDC. The detailed algorithm is given in appendix A.1.

3.1.3 Third Level Deadlock

Definition 1.4: P_{TD} is a set of products without 1st and 2nd level DL. All the movable products in P_{TD} form a set of products $P_M \subseteq P_{TD}$ and P_{SM} is a non-empty subset of P_M. If moving any product in P_{SM} will cause 2nd level DL in P_{TD} and moving any product belonging to P_M- P_{SM} will cause 1st level DL in P_{TD}, then we define P_{TD} as **third level deadlock.**

For convenience of 2nd level DL analysis, we define potential deadlock chain. Thus, in order to well introduce the structure of third level deadlock, we give the following definition:

Definition 1.5: For any second level deadlock P_{SD}, $P_{SD'} = P_{SD} - \{p_i\}$ and p_i is any product belonging to P_{SD}, then we define $P_{SD'}$ as **potential second level deadlock (PSLD)**.

Same as PDC, we should also define the start and end product of PSLD. For the start product, first we check whether there is a product $p_{i'} \in P_{SD'}$ that $HR(p_{i'}) = FRR(p_i)$, if positive, we define $p_{i'}$ as the start product of PSLD, otherwise, we find a product $p_{i''} \in P_{SD'}$ that $HR(p_{i''}) = SRR(p_i)$ and define $p_{i''}$ as the start product of PSLD. For the end product, first we check whether there is a product $p_{j'} \in P_{SD'}$ that $HR(p_i) = FRR(p_{j'})$, if positive, we define $p_{j'}$ as the end product of PSLD, otherwise, we find a product $p_{j''} \in P_{SD'}$ that $HR(p_i) = SRR(p_{j''})$ and $FRR(p_{j''})$ is not occupied by any product in $P_{SD'}$, then we define $p_{i''}$ as the end product of PSLD.

Based on the definition of PSLD, we give two simple instances of 3rd level DL extended from the 2nd level DL shown in Fig. 3.4(a) and (b). Removing product p_4 from the 2nd level DL in Fig. 3.4(a), we get a PSLD $\{p_1, p_2, p_3\}$. p_3 and p_1 are the start and end product of PSLD respectively. If we have another PDC couples with this PSLD in a common idle resource, we will get a 3rd level DL as shown in Fig. 3.6(a). Similarly, if we remove the product p_1 from the 2nd level DL in Fig. 3.4(b), we will get a PSLD $\{p_0\}$ and p_0 is both the start and end product of

PSLD. If we have a simplest PDC couples with this PSLD in a common idle resource, we will get the simplest 3rd level DL as shown in Fig. 3.6(b).

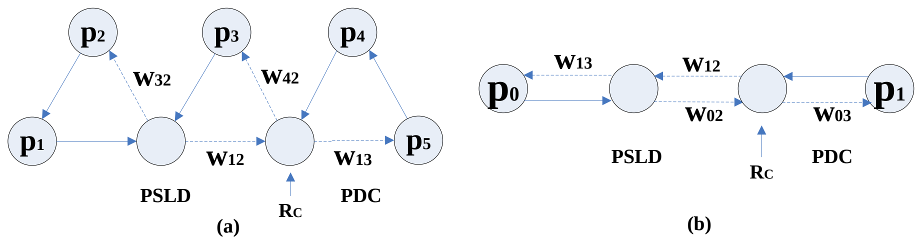


Figure 6. Instances of 3rd Level Deadlock

In Fig. 3.6(a), PSLD {p_1, p_2, p_3} couples with PDC {p_4, p_5} in resource R_C (SRR(p_1) = FRR(p_4) = R_C, HR(p_3) = SRR(p_4) and HR(p_5) = TRR(p_1)). Moving p_1 or p_4 will cause 2nd level DL and moving p_3 will cause 1st level DL. In Fig. 3.6(b), PSLD {p_0} couples with PDC {p_1} in resource R_C (HR(p_0) = TRR(p_1), FRR(p_0) = SRR(p_1), SRR(p_0) = FRR(p_1) = R_C and TRR(p_0) = HR(p_1)). Moving either of two products will cause 2nd level DL.

There are two general patterns of third level deadlock. The first pattern is a PSLD and n ($n \geq 1$) PDC(s) coupling in a common idle resource.

First, we need to find out the general structure of PSLD, which can be obtained by removing a single product from the general pattern of 2nd level DL shown in Fig.3.5. Since the structure in Fig.3.5 consists of n PDCs and every PDC is symmetrical structurally, we can just remove a product from PDC C_0 and obtain the general pattern of PSLD, as shown in the left part of Fig.3.7. Then coupling the PSLD and n ($n \geq 1$) PDC(s) we can get the first general pattern of 3rd level DL, as shown in the right part of Fig.3.7.

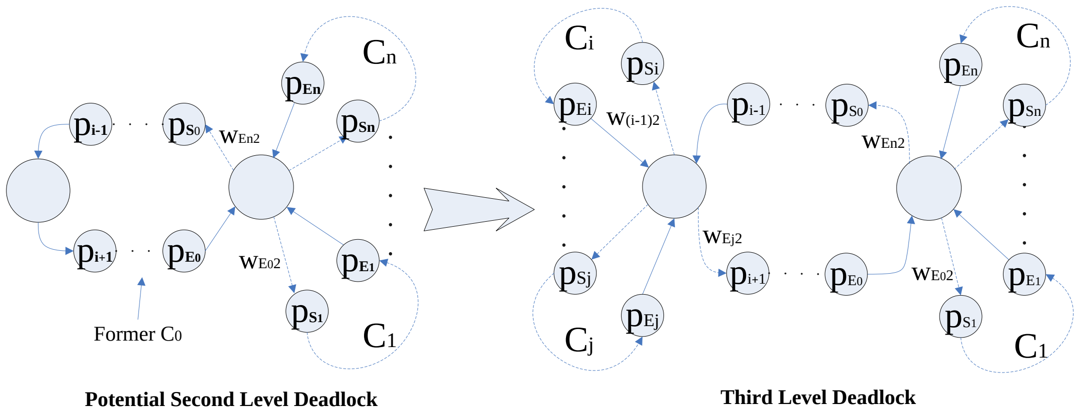


Figure 7. General Pattern of PSLD and 3rd Level DL

In the right part of Fig.3.7, the movable products are the end product of PDCs. Among these movable products, except p_{E0} and p_{i-1}, moving any other product will cause 2nd level DL.

The second pattern of 3rd level DL is caused by the special routine of products, as shown in Fig.3.8. In Fig.3.8, the only movable product is p_{S0}, and after moving p_{E0} we will get the general pattern of 2nd level DL as shown in Fig.3.5.

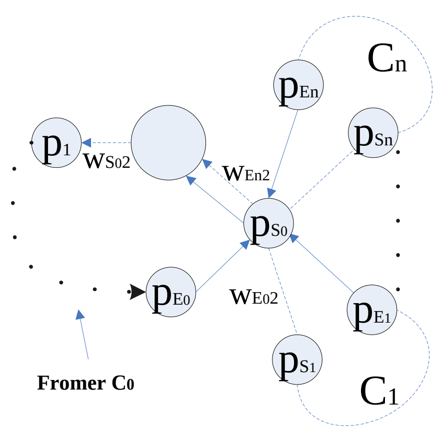


Figure 8. General Pattern of 3rd Level DL

In sum, the two patterns shown in Fig.3.7 and Fig.3.8 cover all the cases of 3rd level DL.

For the first pattern in Fig.3.7, similar to the detection of 2nd level DL, one by one, we assume every product in the system as the end product of a PDC and check whether there is a third level DL from that PDC. The detailed detection algorithm of this pattern is given in appendix A.3.

For the second pattern in Fig.3.8, since there is only one movable product, we can virtually move every product in the system one by one and check whether there is a second level DL in the generated virtual state. Since the detection algorithm of this pattern is almost the same as that of 2nd level DL, we do not introduce the detailed algorithm again.

3.2 Deadlock Detection based on Ranking Matrix

In this section, let us consider a job shop system consisting of N resources, denoted by the set $R=\{r_j, j\in J=\{1, 2,\ldots, M\}\}$, on which M jobs, denoted by the set $P=\{p_i, i\in I=\{1, 2,\ldots, N\}\}$, are processed as they advance through the system. Furthermore, each job holds a resource in an exclusive mode, and follows a predefined working procedure. Through this paper, $S=[s_{ij}]_{M\times N}$ is utilized to characterize the system state at any time point, where s_{ij} represents the number of steps before job p_i being processed on resource r_i. Here $s_{ij}>0$ implies that job p_i has yet been processed on resource r_i; $s_{ij}=0$ implies that job p_i is holding resource r_i; $s_{ij}<0$ implies that job p_i has already been processed on resource r_i. Under a state $S=[s_{ij}]$, if job p_k is moved from the current resource to the next resource, the following state matrix can be derived by $\bar{S}=[\bar{s}_{ij}]$ where $\bar{s}_{ij}=s_{ij}$ if $i\neq k$ and $\bar{s}_{ij}=s_{ij}-1$ if $i=k$. Hence, for the previous example, if we move job p_1 from resource r_2 to r_4, the system state S_1 becomes system state S_2

$$S_1=\begin{bmatrix} 3 & 0 & 2 & 1 \\ 0 & 1 & -1 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix} \xrightarrow{p_1:\ r_2\rightarrow r_4} S_2=\begin{bmatrix} 2 & -1 & 1 & 0 \\ 0 & 1 & -1 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix}$$

3.2.1 First Level Deadlock

Suppose $S=[s_{ij}]_{M\times N}$ represents the system state matrix with $i\in I=\{1,\ 2,\ldots,M\}$ and $j\in J=\{1,\ 2,\ldots,N\}$ representing the job and resource index, respectively.

Definition 2.1 *If there exists such a set of jobs $A=\{a_i,\ i=1,\ \ldots,M_A\}$ in the system that,*

(1) for $i=1$, there does not exist such $b\in I$ and $x\in J$ that $s_{a_i x}=0$ and $s_{bx}=1$.

(2) for $1<i<M_A$, there exists such $y\in J$ that $s_{a_i y}=0$ and $s_{a_{i-1}y}=1$.

(3) for $i=M_A$, there exists such $z\in J$ that $s_{j_k z}=1$ and $\prod_{\forall h\in I} s_{hz}\neq 0$.

then we define a_1 as the tail job of $\forall a_i\in A$, denoted by $TJ(a_i)=a_1$, a_{M_A} as the head job of $\forall a_i\in A$, denoted by $HJ(a_i)=a_{M_A}$, and resource z as the head resource of $\forall a_i\in A$, denoted by $HR(a_i)=z$.

This definition characterizes a set of sequential jobs in which, except the tail and head jobs, every job is holding the resource requested by its predecessor while requesting the resource held by its successor. Moreover, no job in the system is requesting the resource held by the tail job, nor holding the resource requested by the head job. Figure 1 shows an example, in which each rectangle represents a resource, indexed by the symbol on the right top corner. The symbol in the center of the resource represents the job that is holding the resource with blank indicating an available resource. The arrow represents the transition of a job, characterized by the text above it. In the text, the symbol before the colon represents the job; the symbol after the colon but before the arrow represents the original resource; the symbol after the arrow represents the destination resource. These drawing criteria will be utilized throughout the rest of the paper without further clarification. In Figure 3.9, job 1, job 3, and resource 4 are the tail job, head job, and head resource for all the jobs, respectively, i.e., $TJ(1)=TJ(2)=TJ(3)=1$, $HJ(1)=HJ(2)=HJ(3)=3$, and $HR(1)=HR(2)=HR(3)=4$.

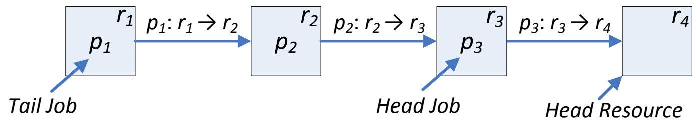


Figure 9. Tail Job, Head Job, and Head Resource

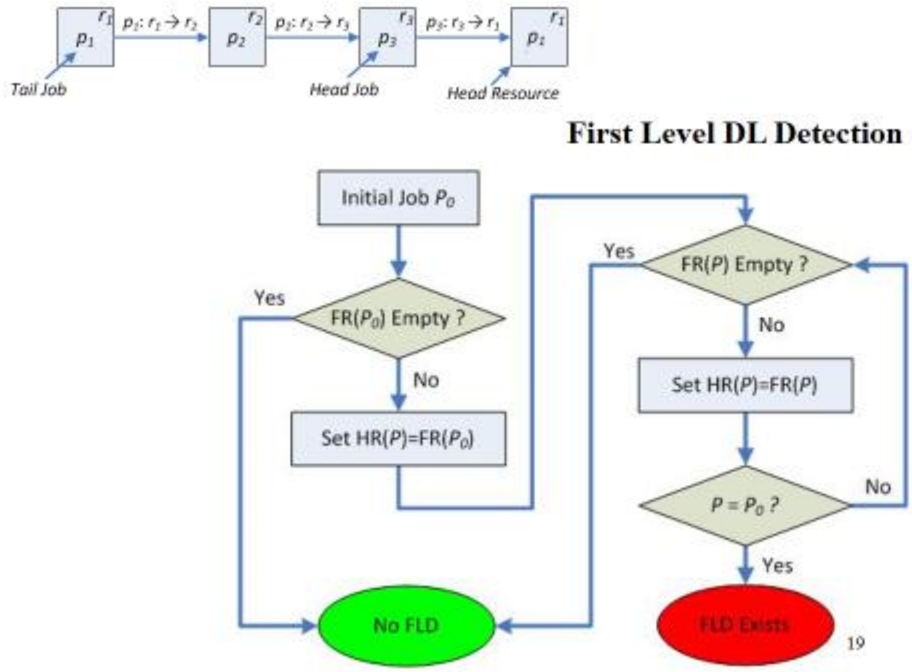


Figure 10. First Level DL Algorithm Flow Chart

Furthermore, we define $TJ(i)=0$, $HJ(i)=0$, and $HR(i)=0$ if the tail job, head job, and head resource do not exist for job i; define $HR(i)=\infty$ if job i is leaving the system immediately as the system output can be considered as a fictitious resource r_∞ with unlimited capacity [2].

Definition 2.2 *The first level DL is a set of jobs in which every job is requesting a resource being held by another job in the set.*

The first level DL defined here is the deadlock defined in other articles [1], which implies an infinite waiting loop in the system. By Definition 1, we propose the following sufficient and necessary condition of the first level DL in Proposition 1.

Proposition 1 *The sufficient and necessary condition of the existence of the first level DL is that there exists such* $a \in I$ *that* $HJ(a) = 0$ (*or* $TJ(a) = 0$ *or* $HR(a) = 0$).

This proposition entails the existence of circular waiting jobs.

3.2.2 Second Level Deadlock

Definition 2.3 *The second level DL is a set of jobs in which*

(a) moving any movable job will lead to a first level DL involving that job and other jobs from the same set.

(b) there does not exist a proper subset of jobs which satisfy condition (a) and (b).

Second level DL has been addressed in the previous articles. Figure 2 shows an example second level DL. Here, for the sake of readability, we utilize dash line to represent the future transition while the solid is still representing the immediate transition of a job. In Figure 2, job 1 is not movable; moving job 2 will lead to a first level DL {2,3}; moving job 3 will lead to a first level DL {1,2,3}. To generalize the structure in Figure 3.10, we propose the following sufficient and necessary condition of second level DL in Proposition 2.

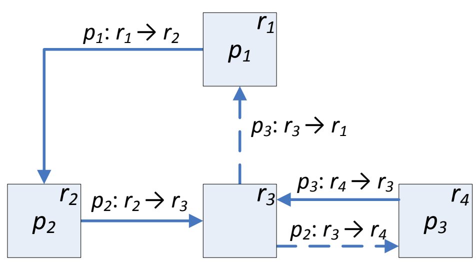


Figure 11. Second Level Deadlock with 4 Resources and 3 Jobs

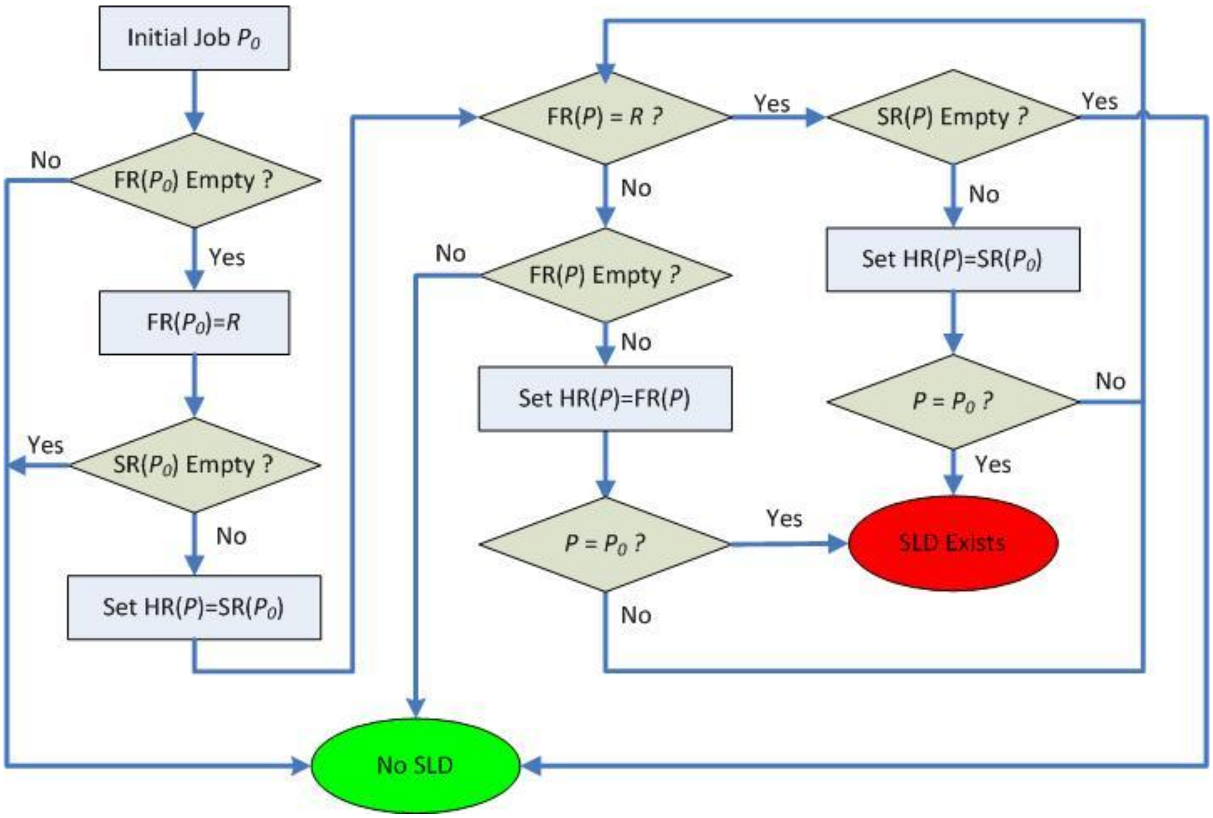


Figure 12. Second Level Deadlock Detection

Proposition 2 *The sufficient and necessary condition of the existence of second level DL is that there exists such* $I_S \subseteq I$ *that*

(a) there exists such $k \in J$ *that, for* $\forall a \in I_S$, $HR(a) = k$.

(b) for $\forall b \in I_S$ *where* $HJ(b) = b$, *there exist such* $c \in I_S$ *and* $\ell \in J$ *that* $s_{b\ell} = 2$, $s_{c\ell} = 0$, *and* $HJ(c) \neq b$.

In Proposition 2, condition (a) implies that all the jobs in the set share the same head resource *k* whereas condition (b) implies that, for any head job *b* in the set, it cannot leave the system in two steps, and the resource it requests in two steps is being held by another job in the set, whose head job is not job *b*. We now apply Proposition 2 to the previous example in Figure 2. The state matrix of the system in Figure 2 is

$$S = \begin{bmatrix} 0 & 1 & X & X \\ X & 0 & 1 & 2 \\ 2 & X & 1 & 0 \end{bmatrix}$$

Here X represents the information which is irrelevant to the detection of the second level DL under the current system state. Obviously, $HR(1) = HR(2) = HR(3) = 3$ (condition (a) in Proposition 1 is satisfied); $HJ(2) = 2$, $s_{24} = 2$, $s_{34} = 0$, and $HJ(3) = 3 \neq 2$; $HJ(3) = 3 \neq 2$, $s_{31} = 2$, $s_{11} = 0$, and $HJ(1) = 2 \neq 3$ (condition (b) in Proposition 1 is satisfied). Therefore, there is a second level DL in current system state.

3.2.3 Third Level Deadlock

Definition 2.4 *The third level DL is a set of jobs in which*

(a) moving any movable job will lead to either a first level DL involving that job and other jobs from the same set, or a second level DL involving that job and other jobs from the same set.

(b) there does not exist a proper subset of jobs which satisfy condition (a) and (b).

Figure 3 shows an example third level DL. In Figure 3, job 1 is not movable; moving job 2 leads to a second level DL {2,3,4}; moving job 3 leads to a second level DL {1,2,3,4}; moving job 4 leads to a first level DL {1,2,4}. To generalize the structure in Figure 3.11, we have the following sufficient and necessary condition of the third level DL in Proposition 3.

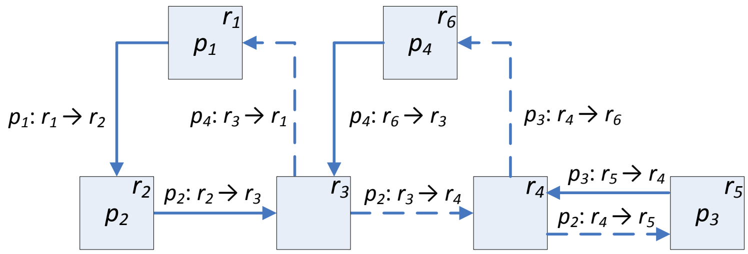


Figure 13. Third Level Deadlock with 6 Resources and 4 Jobs

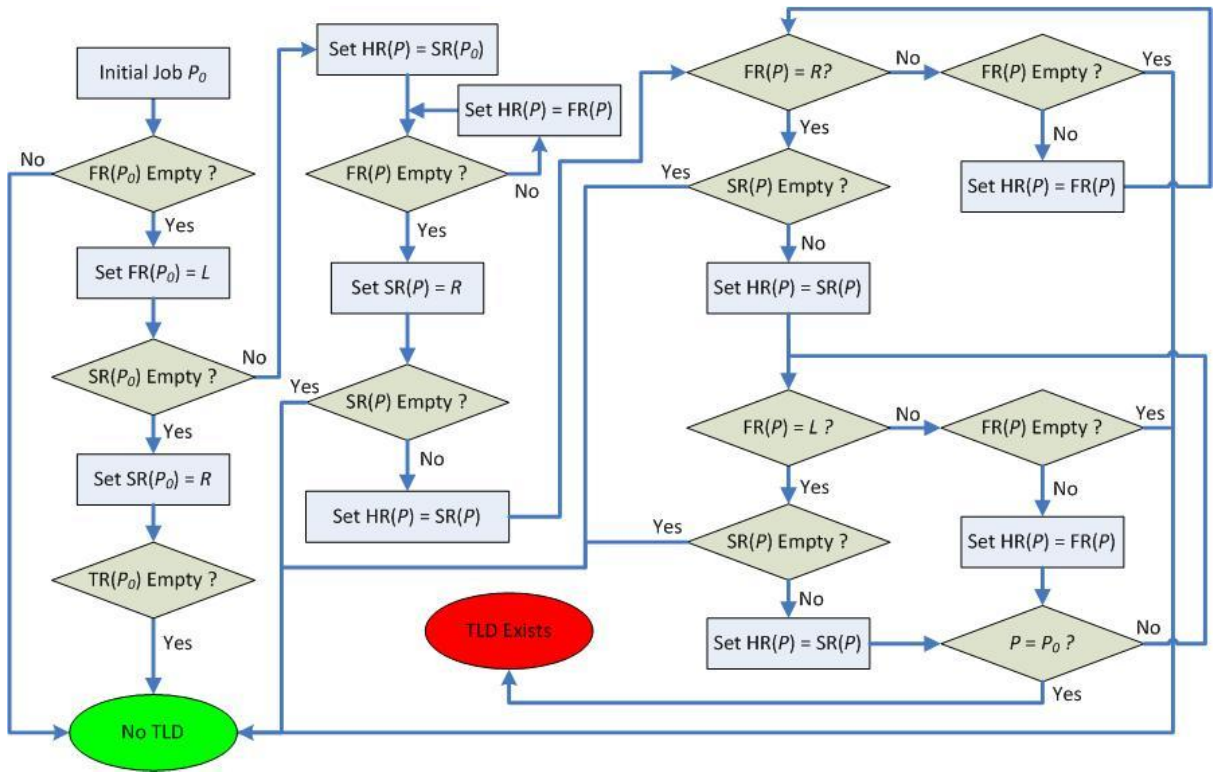


Figure 14. Third Level DL detection

Proposition 3 *The sufficient and necessary condition of the existence of the third level DL is that there exists such* $I_T \subseteq I$ *that*

(a) there exist such $x, y \in J$ *that*

(i) for $\forall a \in I_T$, $[HR(a)-x]\cdot[HR(a)-y]=0$

(ii) $\sum_{\forall a \in I_T}[HR(a)-x]\cdot \sum_{\forall a \in I_T}[HR(a)-y] \neq 0$.

(b) for $\forall b \in I_T$ *where* $HJ(b)=b$

(i) there must exist $k \in J$ *where* $s_{bk}=2$;

(ii) if there exists $c \in I_T$ *where* $s_{ck}=0$, *then* $HJ(c) \neq b$; *otherwise,* $(k-x)(k-y)=0$, *and there must exist* $d \in I_T$ *and* $\ell \in J$ *where* $s_{b\ell}=3$, $s_{d\ell}=0$, *and* $HR(d) \neq HR(b)$.

In Proposition 3, condition (a) implies that there are two head resources (available resources) in the third level DL whereas condition (b) implies that, (i) for any head job *b* in the set, it cannot leave the system in two steps; (ii) if the resource job *b* requests in two steps is being held by another job *c* in the set, the head job of job *c* cannot be job *b*; (iii) if the resource job *b* requests in two steps is not being held by another job in the set, then job *b* cannot leave the system in three steps, the resource job *b* requests in two steps must be one of the head resources, and the resource job *b* requests in three steps is being held by another job *d* in the set, whose head resource cannot be job *b*'s head resource. We now apply Proposition 3 to the previous example in Figure 3. The state matrix of the system in Figure 3 is

$$S = \begin{bmatrix} 0 & 1 & X & X & X & X \\ X & 0 & 1 & 2 & 3 & X \\ X & X & X & 1 & 0 & 2 \\ 2 & X & 1 & X & X & 0 \end{bmatrix}$$

1. Condition (a).(i) & (ii) in Proposition 3 are satisfied as $HR(1) = HR(2) = HR(4) = 3$ and $HR(3) = HR(5) = 4$.
2. Condition (b).(i) in Proposition 3 is satisfied as $HJ(2) = 2$, $HJ(3) = 3$, $HJ(4) = 4$, and $s_{24} = s_{36} = s_{41} = 2$.
3. Condition (b).(ii) in Proposition 3 is satisfied as
   - ◊ Resource 4, which is requested by job 2 in two steps, is available; Also, $s_{25} = 3$, $s_{35} = 0$, and $HR(2) = 3 \neq HR(3) = 4$.
   - ◊ $s_{36} = 2$, $s_{46} = 0$, and $HJ(3) = 3 \neq HJ(4) = 4$.

◊ $s_{41} = 2$, $s_{11} = 0$, and $HJ(4) = 4 \neq HJ(1) = 2$.

Therefore, there is a third level DL in the current system state.

Thus far, we have defined the first, second, and third level DL as well as the corresponding sufficient and necessary conditions. By checking these conditions, we are able to detect whether, under a certain system state, there exists an immediate DL or a DL-free unsafe state [1] (i.e., DL is not avoidable in the future. In the following section, we will apply these DL detection methods to the job shop scheduling problems, and propose a DL-free scheduling algorithm.

CHAPTER IV. POLICY BASED REINFORCEMENT LEARNING ALGORITHM

Since the detection strategy for DL higher than third-level has not been found until now, our DL-free scheduling algorithm is applied off-line before the production batch entering the shop-floor.

For a job-shop system, an efficient DL-free schedule satisfies: (i) jobs are processed based on the predefined routines; (ii) system evolves as jobs advance through the resources without DL occurring, and (iii) resource utilization is maintained in a reasonable level, or in other words, DL-free schedule performance such as makespan is acceptable under the demand constraints. Furthermore, computation time consumed to generate the DL-free schedule should be within an acceptable range so that this algorithm is applicable for real-time implementation.

In this section, suppose that the job-shop system discussed (i) is buffer-less, (ii) consists of M jobs and N resources (every job has N process steps), and (iii) the current system state can be observed. Furthermore, suppose S_0 and S_E represent the initial state (i.e., before each job entering the system) and the final state (i.e., all jobs finished their process), respectively. Hence, there are $M \times N$ (number of jobs $\times$ number of process steps) process steps or $M \times N$ different system states between S_0 and S_E. Therefore, finding a DL-free schedule is actually finding a multi DL-free system states transition route from S_0 to S_E.

Normally, task of reinforcement learning algorithm is to maximize the long-term reward. The Policy search based RL scheduling algorithm introduced in this section aims to find the schedule for buffer-less job-shop system with an optimal or sub-optimal makespan (minimum makespan) effectively. The key point of policy search based multi-state reinforcement learning is to have independent searching action at each state to improve their local policies subject to a common goal. Considering an $M \times N$ JSP problem as a stochastic problem with $M \times N$ states, we choose every operating action through policy search, and action with the best policy performance will be selected. Policy search algorithm searches directly in the space of job shop scheduling dispatching rules without learning value functions, which can be highly effective. Intuitively, at any time point, the higher degree of concurrency, the less makespan the DL-free schedule will be subect to. Motivated by this intuition, we utilize policy based reinforcement learning approach as illustrated below.

Definition 3.1 *Given* $s_i \subseteq S$ *be the current state of operation i, where* $s_i = \{A(\pi_1), A(\pi_2)..A(\pi_i)\}$, { A_i } *corresponds to the set of actions current state can execute,* π_i *corresponding policies of each feasible action. For each action* A_i *at current state, we have a corresponding local action reward parameter* $R(\ell_i)$. *where*

$$R(\ell) = \frac{\sum_{i,k=1}^{n}(J_i + M_k)}{C_{MAX}}$$

$C_{\max}$ *is the corresponding makespan of selecting action,* $\sum_{i,k=1}^{i}(J_i + M_k)$ *is the count summary of jobs and machines been involved during this makespan.*

Here $P(\pi)$ represents the performance of policy π , and $R(\ell)$ represents the local action reward parameter.

Definition 3.2 *Given a distribution over a set* $S_0 \subseteq S$ *of starting states, for policy* π *, which need to be chosen from the spaces of policies at each state until reaching a final state, the performance* $P(\pi)$ *of* π*, is defined as the expected value of local action reward parameter* $R(\ell)$.

$$P(\pi) = E[R_0 \mid s_0 \in S_0, \pi] = E[\sum_{i=1}^{n}(\gamma^i R(i)) \mid \pi] = \lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} R(\ell)$$

In JSP problems, the discount factor γ can be safely set to 1 because the existence of a termination state S_E is assumed. Once the final state is reached, the process stops and receives no further rewards. The essential idea of the policy search-based reinforcement learning is to directly adapt the most promising policy to be learned with respect to its performance, Definition 6 provides a straightforward implementation of a procedure that is tailored for policy search reinforcement learning and that enables a single agent interact with the environment and improve its policy independently.

As discussed before, we consider teams of cooperative actions that all seek to optimize a global reward and we assume that the corresponding multi-action stochastic system can be modeled using our reinforcement learning approach with searching action sets. Therefore, there exists at least one sequence of actions that maximizes the expected reward $P(\pi)$ for all actions, or we can say, minimizes the makespan for JSSP.

There are two strategies when selecting the transition:

(1) DL DETECTION

According to Proposition 1, 2, and 3, by checking the system state matrix, we are able to tell whether an operation/state change will result in first, second, or third level DL. Therefore, a set of DL-free operations can be generated out of the DL detection process.

(2) POLICY BASED REINFORCEMENT LEARNING ALGORITHM

After the DL detection process, a policy based RL algorithm will be applied against the DL-free operations so as to select the operation with the highest priority to move. After moving the highest priority operations to its next resource, the system advances to the next state, and hence, the system state matrix will be updated.

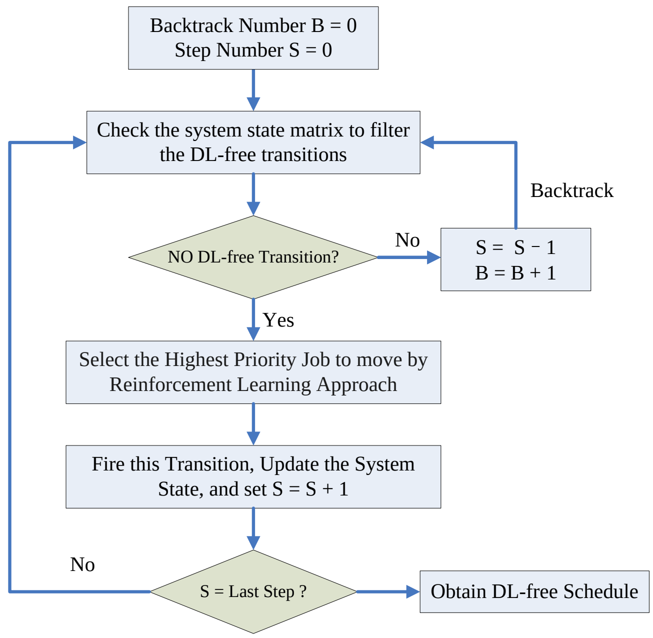


Figure 15. Flow Chart of DL-free Scheduling Algorithm

However, since our DL detection algorithms are not applicable to DL higher than third level, it is possible that, under some system states, no feasible transition is detected to be DL-free. This is because a DL higher than third level has already occurred in the previous system state without being detected. Under this circumstance, we have to "backtrack" the system and choose another DL-free transition with a relatively lower priority. Here by "backtrack", we mean recovering the system to the state one-step before. Therefore, the system will reach the final state S_E in the end and a DL-free schedule will be obtained. The flow chart of DL-free scheduling algorithm is given in Figure 4. The details of the algorithm are given in Appendix A2.

CHAPTER V. EXPERIMENTS

In this section, we apply the DL-free scheduling algorithm to the job-shop benchmark problems. These problems are widely-used as reference for researchers to check the performance of their scheduling methods in terms of makespan, flow time, and/or computation time. For detailed description of these benchmark problems, please refer to [20].

There are 40 benchmark problems of 8 different scales (5 problems for each scale). Data of each problem includes scale (number of jobs and resources), process route of each operation action, and process time for each step. These problems are designed with complicated resource crosscutting structure to increase the scheduling difficulty. Hence, if these systems are assumed to be buffer-less, the scheduling problem will become even more difficult.

We first study the performances of our DL-free scheduling algorithm in terms of makespan and backtrack number. Subsequently, we compare the computational time between scheduling algorithm with and without DL detection. The experimental results of 40 benchmark FMSs scheduling using our DL-free scheduling algorithm are given in Table 1.

Table 1. Results of 40 benchmark job shop instances

Problem (size)	2LD		3LD		Makespan	Optimal Makespan
	Time	Backtrack	Time	Backtrack		
FT06	<1 sec.	0	<1 sec.	0	512	512
LA01 (10x5)	<1 sec.	0	<1 sec.	0	1096	1073
LA02 (10x5)	<1 sec.	0	<1 sec.	0	1025	1025
LA03 (10x5)	<1 sec.	0	<1 sec.	0	857	817
LA04 (10x5)	<1 sec.	0	<1 sec.	0	933	827
LA05 (10x5)	<1 sec.	0	<1 sec.	0	879	879

LA06 (15x5)	<1 sec.	0	<1 sec.	7	1390	N/A in 48 hours
LA07 (15x5)	<1 sec.	0	<1 sec.	13	1337	N/A in 48 hours
LA08 (15x5)	<1 sec.	0	<1 sec.	19	1314	N/A in 48 hours
LA09 (15x5)	<1 sec.	0	<1 sec.	0	1609	N/A in 48 hours
LA10 (15x5)	<1 sec.	0	<1 sec.	8	1525	N/A in 48 hours
LA11(20x5)	<1 sec.	0	<1 sec.	3	1873	N/A in 48 hours
LA12 (20x5)	<1 sec.	0	<1 sec.	17	1726	N/A in 48 hours
LA13 (20x5)	<1 sec.	0	<1 sec.	14	1895	N/A in 48 hours
LA14 (20x5)	<1 sec.	0	<1 sec.	0	1901	N/A in 48 hours
LA15 (20x5)	<1 sec.	0	<1 sec.	0	2015	N/A in 48 hours
LA16 (10x10)	<1 sec.	1	<1 sec.	16	1498	N/A in 48 hours
LA17 (10x10)	<1 sec.	0	<1 sec.	113	1187	N/A in 48 hours
LA18 (10x10)	<1 sec.	0	<1 sec.	12	1478	N/A in 48 hours
LA19 (10x10)	<1 sec.	6	<1 sec.	31	1412	N/A in 48 hours
LA20 (10x10)	<1 sec.	0	<1 sec.	0	1514	N/A in 48 hours
LA21 (15x10)	<1 sec.	21	<1 sec.	409	2051	N/A in 48 hours
LA22 (15x10)	<1 sec.	29	3 sec.	12053	1811	N/A in 48 hours
LA23 (15x10)	<1 sec.	143	<1 sec.	1339	2032	N/A in 48 hours
LA24 (15x10)	<1 sec.	22	<1 sec.	888	1934	N/A in 48 hours
LA25 (15x10)	<1 sec.	108	<1 sec.	12430	1983	N/A in 48 hours
LA26 (20x10)	<1 sec.	38	45 sec.	624349	2666	N/A in 48 hours
LA27 (20x10)	<1 sec.	36	<1 sec.	221	2730	N/A in 48 hours
LA28 (20x10)	<1 sec.	6	<1 sec.	799	2600	N/A in 48 hours
LA29 (20x10)	<1 sec.	196	<1 sec.	8683	2621	N/A in 48 hours
LA30 (20x10)	<1 sec.	23	<1 sec.	591	2774	N/A in 48 hours

LA31 (30x10)	<1 sec.	33	<1 sec.	3174	3701	N/A in 48 hours
LA32 (30x10)	<1 sec.	73	47 sec.	295725	3997	N/A in 48 hours
LA33 (30x10)	<1 sec.	71	4 sec.	46982	3791	N/A in 48 hours
LA34 (30x10)	<1 sec.	94	4 sec.	26426	3929	N/A in 48 hours
LA35 (30x10)	<1 sec.	68	<1 sec.	9705	4076	N/A in 48 hours
LA36 (15x15)	<1 sec..	69	Not available in 3 hrs		2543	N/A in 48 hours
LA37 (15x15)	<1 sec.	239	<1 sec	1765	2800	N/A in 48 hours
LA38 (15x15)	<1 sec.	339	8 min.		2301	N/A in 48 hours
LA39 (15x15)	<1 sec.	35	<1 sec.	1922	2386	N/A in 48 hours
LA40 (15x15)	<1 sec.	415	15 min.	8518357	2578	N/A in 48 hours

From the results shown in Table 1, we have the following explanations and conclusions:

(1) All benchmark problems can be solved via using our DL-free scheduling algorithm within reasonably acceptable time. For small- and middle-size problems, the search time is less than 1 second.

(2) The obtained makespans are for small-size job shop problems are very close to the optimal value. We notice that, for some problems such as FT06, LA02, and LA05, the obtained makespans are actually optimal. These results imply that the policy based reinforcement learning approach is extremely efficient in terms of reducing makespan. We also notice that, due to the complexity of these problems. The optimal values cannot be found in 48 hours, while our algorithm is able to find the DL-free schedule with sub-optimal makespan within 1 second for all the presented job shop problems.

(3) The total flow time of the obtained DL-free schedules is fairly close to the optimal value for

small-size job shop problems. As for the middle- and large-size job shop problems, the optimal flow time cannot be found in 48 hours. Therefore, the results presented in this paper provide the upper bounds of the total flow time of the DL-free schedules for those job shop problems, which can be utilized as references for future research.

(4) Backtracks number increases as the system becomes larger. For small-scale systems, such as LA01~LA15, backtracks for most problems are 0 (except LA12 and LA13). This is because, for systems with 5 resources, each operation action has only 5 processing steps. Hence, it is rare to generate a system state with DL higher than third level (recall that DL lower than forth level can be detected) during the manufacturing process, even in the well-designed benchmark problems. For large-scale FMSs, more work-pieces/jobs and longer processing routes may cause the occurrence of DL higher than third level, which results in Backtrack. However, the number of backtrack used is kept in a relatively low level for all benchmark problems.

Therefore, one may expect our DL-free scheduling algorithm to be applied to other problems with similar scale and structure features. Moreover, although this is an off-line scheduling algorithm, the negligible computation time makes it an eligible tool for quick response to the disturbances in systems, such as adding new JOBs, removing JOBs being processed, change of priorities, and machine breakdown. Once a disturbance event happens, our scheduling algorithm can re-schedule and generate a new DL-free schedule within 1 second, which is indistinguishable to the on-line scheduling.

According to DL-free schedules obtained, one may picture the Gantt chart of production process. To increase the intuitiveness of job shop scheduling, as an example, Figure 5 and 6 give the Gantt chart of LA08 (scale of 15×5) and LA16 (scale of 10×10), which can be used as references for real shop implementation. In Figure 5 and 6, each row represents the loading process of a machine; each block represents a single process on a machine; block width represents the process time; the number at top left corner of each block represents the operation action label; the number at top left corner of the whole figure is the makespan. Reader may check the process and prove the schedule is DL free.

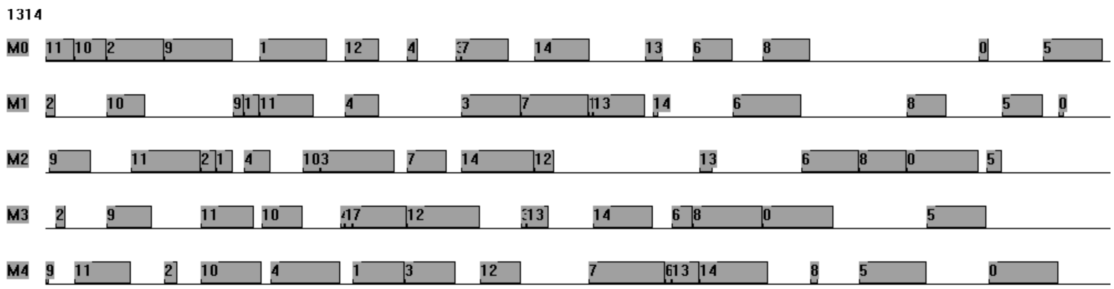


Figure 16. DL-free Schedule of LA08 (15 jobs on 5 machines) with makespan = 1314

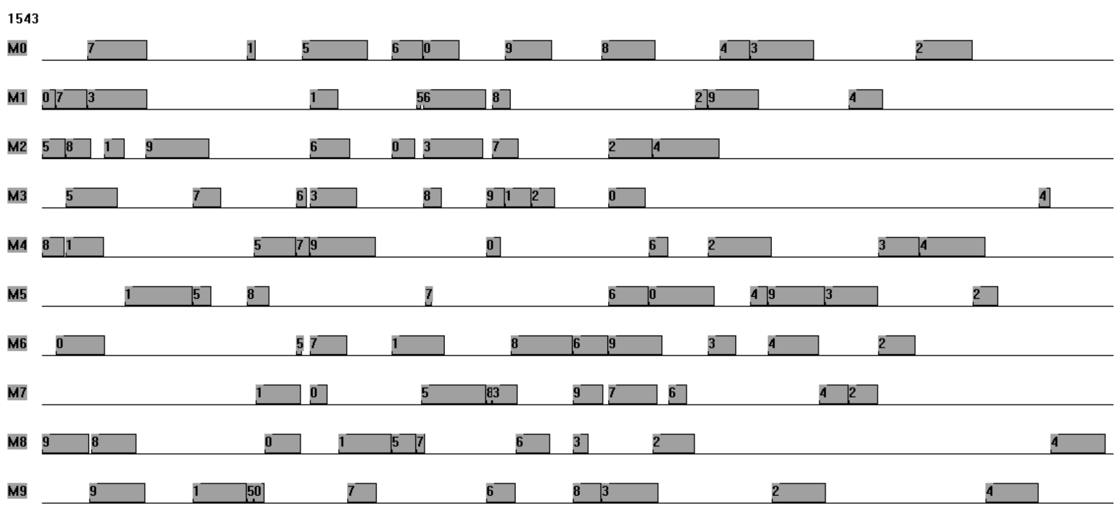


Figure 17. DL-free Schedule of LA16 (10 jobs on 10 machines) with makespan = 1543

CHAPTER VI. CONCLUSION AND FUTURE RESEARCH

In this paper, based on transition graph model, we analyzed the general pattern of high level DL in the context of discrete manufacturing systems, and propose the corresponding detection algorithms. With the help of these DL detection algorithms, we were able to develop a DL-free scheduling algorithm for buffer-less job shop FMSs (e.g., production plants for air-plane head or turbine dynamo shaft). Applying our DL-free scheduling algorithm to 40 widely-used benchmark problems, DL-free schedules for all the benchmark FMSs can be obtained within 1 second, with acceptable makespans. To the best of our knowledge, for FMSs with considerable scale, under the restriction of NO-buffer, those solutions have never been published in the extant literature.

Hence, it is safe to conclude that our DL-free scheduling algorithm is applicable to the real implementations of job-shop FMSs. Furthermore, the basic idea of our high level DL detection algorithm can also be extended to other man-made "resource intensive and highly sharing" systems (e.g., computer operation systems, communication systems, and traffic systems), which could be one direction of the future works.

APPENDIX

Representation Scheme:

M	number of Jobs in the system
N	number of resources in the system
S	number of system state in a search
unblk[S]	Set of unblocked (movable) job(s) at step *S*
DLfr[S]	Set of DL-free job(s) at step *S*
HR(p_j)	the resource which job p_j is holding
FRR(p_j)	the resource which job p_j requests next step
SRR(p_j)	the resource which job p_j requests two steps later
TRR(p_j)	the resource which job p_j requests three steps later
S_E	final state of the system

A.1 Deadlock-free Scheduling Algorithm

The details of our deadlock free scheduling algorithm are given as follows, and readers may refer to the flow chart of the schedule algorithm shown in Fig.3.1:

Step 1: System initialization. Set search index SI = 0

Step 2: Search initialization. Set step number S = 0, backtrack number B = 0

Step 3: From the unblocked products set of current step P_unblk[S], using DL detection algorithm, we exclude the product(s) which will cause deadlock or DL free unsafe state, and put the left products(s) into DL free products set of current step P_DLfr[S]. If P_DLfr[S] Ø, go to step 5. Otherwise, set B = B + 1 and check whether B reaches UBB. If B = UBB, go to step 5; if B<UBB, recover the system to step S-1 and set S = S – 1, then try Step 3 again.

Step 4: Use HNGP to select one product p from P_DLfr[S], set P_unblk[S] = P_unblk[S] – {p}. Move product p, set S = S + 1 and update the state of system. If S = SE, end algorithm and store the obtained schedule.

Step 5: Set SI = SI + 1, if SI reaches the product number M, end the program. Otherwise, go back to step 2 and start another search.

A.2 Second Level Deadlock Detection Algorithm

Assume that the set of products processing in the system is P= {pj | j = 1, …, M} and set j=1 at the beginning of the algorithm. Now we describe the 2nd level deadlock detection algorithm as follows:

If j>M, there is no 2nd level DL and we end the algorithm. If pj doesn't have two more steps to process in future or SRR(pj) is idle, set j = j+1 and repeat (1); otherwise, we use pT to denote the product that occupies SRR(pj) and set FRR(pj) as central resource RC, then go to (2)

If FRR(pT) = RC, then go to (3); otherwise go to (4)

If SRR(pT) = HR(pj), then the system is under 2nd level deadlock status and we end the algorithm; if SRR(pT) is idle or the product processing in SRR(pT) has already been searched (that is have already been denoted by pT), then we set j=j+1 and go back to (1); else we use pT to denote the product processing in SRR(pT) and go back to (2)

(4) If FRR(pT)= HR(pj) and the algorithm has at least been to step(3) once, then the system is under 2nd level deadlock status and end the algorithm; if FRR(pT) is idle or the product processing in FRR(pT) has already been searched, then we set j=j+1 and go back to (1); else we use pT to denote the product processing in FRR(pT) and go back to (2)

The algorithm is in polynomial computation, since only M products in system and utmost N-1 steps may be tried to find the conclusion (usually M is bigger than N), the computation complexity is O(1)->O(N2 - 2N + 1).

A.3 Third level Deadlock Detection Algorithm

Assume the set of products processing in the system is P= {pj | j = 1, …, M}, where N<M, N is the number of machine resource in system. Set j=1 at the beginning of the algorithm. Now we describe the algorithm to detect the first pattern of 3rd level deadlock shown in Fig.3.7 as follows:

If j>M, the system is not under third level deadlock and end the algorithm, then let j=j+1 and repeat (1). If pj does not have three more steps to process in the future or TRR(pj) is idle, then let j=j+1 and go to (1), else use p to denote the product that occupies TRR(pj) and set FRR(pj) and SRR(pj) as FCR and SCR respectively, then go to (2)

If FRR(pT)=SCR, then go to (3); otherwise, go to (6)

If SRR(pT)=FCR, then go to (4); otherwise, go to (5)

If TRR(pT)= HR(pj), then system is under 3rd level deadlock and terminate the algorithm; if TRR(pT) is idle or the product processing in TRR(pT) has already been searched, then set j=j+1 and go to (1); otherwise, set pT to be the product processing in TRR(pT) and go to (7)

If SRR(pT) is idle or the product processing in SRR(pT) has already been searched, then let j=j+1 and go to (1); otherwise, set pT to be the product processing in SRR(pT) and go to (2)

If FRR(pT) is idle or the product processing in FRR(pT) has already been searched, or FRR(pT)= HR(pj), then let j=j+1 and go back to (1), otherwise, set pT to be the product processing in FRR(pT) and go to (2)

If FRR(pT)=FCR, then go to (8); otherwise, go to (10)

If SRR(pT)= HR(pj), then the system is under 3rd level DL and end the algorithm; if SRR(pT) = SCR, then go to (9); otherwise, set pT to be the product processing in SRR(pT) and go to (7)

If TRR(pT) is idle or the product processing in TRR(pT) has already been searched, then let j=j+1 and go to (1); otherwise, set pT to be the product processing TRR(pT) and go back to (7)

If FRR(pT)= HR(pj), then the system is under 3rd level deadlock and end the algorithm; if FRR(pT) is idle or the product processing in FRR(pT) has already been searched, then let j=j+1 and go back to (1); otherwise, set pT to be the product processing in FRR(pT) and go to (7)

The algorithm is in polynomial computation, since only M products in system and utmost N-1 steps may be tried to find the conclusion (usually M is bigger than N), the computation complexity is O(1)->O((N-1)(N-2)(N-3)).

REFERENCES

[1] Abdallah, I. B., Elmaraghy, H. A., Elmekkawy, T. (2002). Deadlock free scheduling in flexible manufacturing systems using Petri Net, *Int. J. of Prod. Res.*, 40, 2733-2756.

[2] Abdallah, I. B., Elmaraghy, A. (1998). Deadlock prevention and avoidance in FMS: A Petri net based approach, *Int. J. Adv. Manuf. Tech.*, 14(10), 704–715.

[3] Cassandras, C. G., Lafortune, S. (1999). *Introduction to Discrete Event Systems*. Boston, MA.

[4] Cho, H., Kumaran, T. K., Wysk, R. A. (1995). Graph-Theoretic deadlock detection and resolution for flexible manufacturing systems. *IEEE Trans. Robotics Automat.*, 11, 413–421.

[5] Chu, F., Xie, X. (1997). Deadlock analysis of Petri nets using siphons and mathematical programming. *IEEE Trans. Robot. Automat.*, 13, 793–804.

[6] Coffman, E. G., Elphick, M. J., Shoshani, A. (1971). System deadlocks. *ACM Comput. Surv.*, 3, 67–78.

[7] Ezpeleta, J., Colom, J. M., Martinez, J. (1995). A Petri net based deadlock prevention policy for flexible manufacturing system. *IEEE Trans. Robotics Automat.*, 11, 173–184.

[8] Fanti, M. P., Maione, B. , Mascolo, S. , Turchiano, B. (1997a). Event-based feedback control for deadlock avoidance in flexible production systems. *IEEE Trans. on Robotics and Automat.*, 13(3), 347-363.

[9] Fanti, M. P., Maione, B., Mascolo, S., Turchiano, B. (1996). Performance of deadlock avoidance algorithms in flexible manufacturing systems. *J. Manufact. Syst.*, 15(3), 164–178.

[10] Fanti, M. P., Maione, B., Mascolo, S., Turchiano, B. (1997b). Low-Cost deadlock avoidance policies for flexible production systems. *Int. J. Model. Simulation*, 17(4), 310–316.

[11] Fanti, M. P., Zhou, M. C. (2004). Deadlock control methods in automated manufacturing systems. *IEEE Trans. on Systems, Man and Cybernetics, A*, 34(1), 5-22.

[12] Gebraeel, N. Z., Lawley, M. A. (2001). Deadlock detection, prevention and avoidance for automated tool shared systems, *IEEE Trans. on Robotics and Automation*, 17, 342-356.

[13] Gen, M., Cheng, R. (2000). Genetic Algorithms and Engineering Optimization, John Wiley & Sons.

[14] Ghaffari, A., Nidhal, N., Xie, X. L. (2003). Design of live and maximally permissive Petri net controller using the theory of regions, *IEEE Trans, on Robotics and Automation*, 19, 137-142.

[15] Gold, E. M. (1978). Deadlock prediction: easy and difficult cases, *SIAM J. Com.*, 7, 320-336.

[16] Hsieh, F., Chang, S. (1994). Dispatching-driven deadlock avoidance controller synthesis for flexible manufacturing systems. *IEEE Trans, on Robotics and Automation*, 10, 196–209.

[17] Kolonko, M. (1999). Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operation Research*, 113, 123-136.

[18] Kumaran, T. K., Chang, W., Cho, H., Wysk, R. A. (1994). A structured approach to deadlock detection, avoidance and resolution in flexible manufacturing systems. *Int. J. Prod.*

Res., 32(10), 2361–2379.

[19] Lawley, M. A., Reveliotis, S. A., Ferreira, P. (1998). The application and evaluation of Banker's algorithm for deadlock freebuffer space allocation in flexible manufacturing systems. *International Jr. of Flexible Manufacturing Systems*, 10, 73-100.

[20] Lawrance, S. (1984). Resource constrained project scheduling: an experiment investigation of heuristic scheduling techniques, Dissertation, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1984.

[21] Lee, D., DiCesare, E. (1994). Scheduling FMS using Petri net and heuristic search. *IEEE Trans. on Robotics and Automation*, 10(2), 123-132.

[22] Li, Z., Zhou, M. C. (2004). Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems, *IEEE Trans. on System, Man and Cybernetics*, *A*, 34(1), 38-51.

[23] Mattfeld, D. (1995). Evolutionary search and the job shop: investigation on genetic algorithms for production scheduling, Springer.

[24] Park, J., Reveliotis, S. A. (2000). Algebraic synthesis of efficient dead lock avoidance policy for flexible manufacturing systems. *IEEE Trans. on Robotics and Automation*, 16, 190–195.

[25] Ramaswamy, S., Joshi, S. (1996). Deadlock free scheduling for automated manufacturing workstation. *IEEE Trans, on Robotics and Automation*, 12(3), 391-400.

[26] Ramirez, S. A., Benhabib, B. (2000). Supervisory control of multiworkcell manufacturing systems with shared resources. *IEEE Trans. Syst., Man, Cybern*. *part B*, 30, 668–683.

[27] Reingold, E. M., Nievergelt, J., Deo, N. (1997). Combinatorial Algorithms: Theory and Practice. Englewood Cliffs, Prentice-Hall, NJ.

[28] Reveliotis S. A., Ferreira, P. M. (1996). Deadlock avoidance policies for automated manufacturing cells. *IEEE Transactions on Robotics and Automation*, 12(6), 845-857.

[29] Reveliotis, S. A., Lawely, M. A., Ferreira, P. M. (1997). Polynomial-complexity deadlock avoidance policies for sequential resource allocation systems. *IEEE Trans. Automat. Contr.*, 42, 1344–1357.

[30] Shi, X. Q., Wu, Z. M. (2005). Deadlock free scheduling method for FMSs using beam search. *IEEE Inter. Conf. on System, Man and Cybernetics*, 2, 1188-1193.

[31] Taubin, A., Kondratyev, A., Kinshinevsky, M. (1998). Deadlock prevention using Petri nets and their unfoldings. *Int. J. Manufact. Technol.*, 14, 750–759.

[32] Viswanadham, N., Narahari, Y., Johnson, T. L. (1990). Deadlock prevention and deadlock avoidance in flexible manufacturing system using Petri net models. *IEEE Trans. on Robotics and Automation*, 6(6), 713-723.

[33] Wu, N. Q. (1999). Necessary and sufficient conditions for deadlock-free operation in flexible manufacturing systems using a colored Petri net model. *IEEE Trans. Syst., Man, Cyber. C*, 29, 192–204.

[34] Wu, N. Q., Zhou, M. C. (2001). Avoiding deadlock and reducing starvation and blocking

in automated manufacturing systems. *IEEE Trans. on Robotics and Automation*, 17, 657-668.

[35] Wysk, R. A., Yang, N. S., Joshi, S. (1991). Detection of deadlocks in flexible manufacturing cells. *IEEE Transactions on Robotics and Automation*, 7(6), 853-859.

[36] Wysk, R. A., Yang, N. S., Joshi, S. (1994). Resolution of deadlocks in flexible manufacturing systems: Avoidance and recovery approaches. *J. Manufact. Syst.*, 13(2), 128–138.

[37] Xia, W. J., Wu, Z. M. (2006). A hybrid particle swarm optimization approach for the job-shop scheduling problem. *Inter. Jr. Advanced Manufacturing Technology*, 29, 360-366.

[38] Xie, X., Jeng, M. (1999). ERCN-merged nets and their analysis using siphons, *IEEE Trans. on Robotics and Automation*, 15(4), 692-703.

[39] Xiong, Zhou, M. C. (1997). Deadlock-free scheduling of an automated manufacturing system based on Petri nets. *Proceedings-IEEE Inter. Conf. on Robotics and Automation*, 2, 945-950.

[40] Xu, G., Wu, Z. M. (2004). Deadlock free scheduling strategy for automated production cell. *IEEE Trans, Systems, Man and Cybernetics, part A*, 34(1), 113-122.

[41] Yalcin, A., Boucher, T. O. (2000). Deadlock avoidance in flexible manufacturing systems using finite automata. *IEEE Trans. on Robot. Auto.*, 16, 424–429.

[42] Zhou, M. C., Venkatesh, K. (1998). Modeling, Simulation and Control of Flexible Manufacturing Systems, A Petri Net Approach. World Scientific, Singapore.

[43] S. E., Ramaswamy and S. B. Joshi, "Deadlock-free scheduling for automated manufacturing workstations," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 391-400, 1996.

[44] W. Zhang and T. Dietterich, "A reinforcement learning approach to job-shop scheduling," *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. pp. 1114-1120, Montreal, Canada, 1995.

[45] R. Sutton and A. Barto, "Reinforcement Learning. An Introduction," *MIT Press*, Cambridge, MA, USA, 1998

[46] S. Mahadevan N. Marchalleck, T. Das, and A. Gosavi, "Self-improving factory simulation using continuous-time average-reward reinforcement learning," *Proceedings of the 14th International Conference on Machine Learning*, pp. 202-210, Nashville, USA, 1997.

[47] D. Zeng and K. Sycara, "Using case-based reasoning as a reinforcement learning framework for optimization with changing criteria," *Proceedings of the 7th International Conference on Tools with Artificial Intelligence*, pp. 56-62, Takamatsu, Japan, 1997

[48] J. Schneider, W. Wong, A. Moore, and M. Riedmiller, "Distributed value functions," *Proceedings of 16th International Conference on Machine Learning*, pp. 371-378, Beld, Slovenia, 1999.