Deadlock-Detection via Reinforcement Learning

Chen M* and Rabelo L

Department of Industrial Engineering and Management Systems, University of Central Florida, Orlando, USA

Abstract

Optimization of makespan in scheduling is a highly desirable research topic, deadlock detection and prevention is one of the fundamental issues. Supported by what learned from this class, a reinforcement learning approach is developed to unravel this optimization difficulty. By evaluating this RL model on forty classical non-buffer benchmarks and compare with other alternative algorithms, we presented a near-optimal result.

Keywords: Reinforcement learning; Optimization makespan; DL detection

Introduction

Due to buffer-less setting, deadlock (DL) occurs frequently in resource sharing environment and concurrent computing systems. A deadlock is a state in which each member of a group of actions is waiting for some other member to release a lock. [1] Once this DL state occurred, workflow would stack in a fixed loop and never discharged.

Figure 1 present a typical scheduling problem: 3 jobs need to be operated on 3 different machines following different sequences, each machine can only operate one job each time. How to schedule jobs in specific sequence to minimize total makespan aka processing time without deadlock is a typical optimization problem. Due to limitation of resources, deadlock happened frequently, other than a feasible solution, to find the global optimal deadlock free solution is difficult.

There are certain methods to solving deadlock problems: 1. Do nothing, 2. Kill the workflow, 3. Preempt and rollback. Other than kill the workflow, deadlock detection algorithms are more efficient in most cases and additionally, deadlock free scheduler would enable the real-time control for engineering system. Preventing or avoiding deadlock helping maintain system performance aka makespan stay at positive level.

A simple head-tail scheduling example is present in Figure 2. The rectangle stands for resource Ri, symbol Pi stands for jobs, if rectangle is empty them means no job is operating on that resource. Arrow stands for the action of each job. Second level DL has been addressed in the previous articles. Here on Figure 3, a deck lock is presented. Job P3 is moving to resource 3 then resource 1, but once it moved it will be a deadlock because P1 and P2 will be non-moveable. If P2 move to resource 3 first, then P2 and P3 will be a deadlock as they heading to other’s occupation while there is no buffer.

In this paper we present a new reinforcement learning approach solving this scheduling optimization problem. We will test our algorithm’s on the classical buffer-less benchmark and compare with the optimal solution.

Related Work

Local and global deadlock-detection in component-based systems are NP-hard [2]. Wysk et al. [3] developed a deadlock detection via integer programming in finding the optimum makespan. Specifically, they added constraints to ensure agents not release resource unless being assigned the next resource. However, integer programming method can only be used on small size problem as it would take very long time to run. Their integer programming formulation is shown below.

\[ X_{ai}: \text{Completion time of job i (term K);} \]
\[ X_{ki}: \text{Completion time of job i operation k on machine j;} \]
\[ T_{ai}: \text{processing time of job i operation k on machine j;} \]
\[ Y_{jprq}: \{0, 1\}: 1 \text{ if job p operation r follows job q operations on machine j; } 0 \text{ otherwise.} \]

Figure 1: 3 machine 3 job scheduling problem.

Figure 2: Deadlock I level.

Figure 3: Deadlock II level.
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:
\[ \text{Min } Z = \sum_{j} X_{ij} \]
\[ X_{ij} - T_{ik} \geq X_{ik} + T_{kl} \]
\[ X_{ij} - X_{ip} + H(1-Y_{pq}) \geq T_{jq}, [\forall q \in J; \forall (p,q) \in E] \]
\[ X_{ij} - X_{ip} + HY_{ho} \geq T_{ho}, [\forall h \in J; \forall (p,h) \in E] \]
\[ X_{jp} - X_{jq} + H(1-Y_{pq}) \geq E, [\forall q \in J; \forall (p,q) \in E; \forall (q,s) \in E] \]
\[ X_{jp} + HY_{ho} \geq E, [\forall q \in J; \forall (p,q) \in E; \forall (q,s) \in E] \]

In view of the modeling frameworks in the existing literature, three strategies for processing DL and corresponding research work are as follows:

- **Deadlock Prevention**, which organizes resource usage by each process to ensure that at least one process is always able to get all the resources it needs [4]. Mixed integer programming [5] and region theory [6] are used to solve elementary siphon [7-9] to avoid deadlock. Edsger et al., Gen and Cheng [6,10] developed Banker’s algorithm in 2006. However, these algorithms have many limitations: need fixed processing numbers; no further processing can be started quickly deadlock issue once detected. This can be completed in a couple ways [6,14], such as aborting certain action or add additional buffer. Still, detect deadlock and schedule a deadlock free path may be more convenient.

- **Deadlock Avoidance** based on the current system state and agents’ future resource request, by restricting the resource allocation to avoid the deadlock. Petri Net model [3,11-14] are complete developed in this area. Di-graph model and Auto-mata model [6,9,12,15,16] were also built to handle avoidance problems. However, current avoidance algorithms are not able to handle high level DL.

- **Deadlock Detection and Recovery** is more focusing on quickly deadlock issue once detected. This can be completing in a couple ways [6,14], such as aborting certain action or add additional buffer. Still, detect deadlock and schedule a deadlock free path may be more convenient.

Lots of artificial intelligence and operation research effort has been applied to in scheduling problems. Zhang and Dietricher [17] were the first who applied reinforcement learning here. Mahadevan et al. [18] proposed a reinforcement learning algorithm combines different scheduling problem for the optimization of transfer-lines in manufacturing systems. Another maintenance-based approach based on simplified reinforcement learning is suggested by Zeng and Sycara [19].

**Research Methodology**

First let’s give definition of deadlock on different levels. The first level of a DL is a set of agents that each request collects the resources held by another agent. The second-level DL is a set of agents any action will result in a first-level DL. The high level DL is a set of agents moving any action will result in a second-level DL. Figure 4 deliver a graph of how a high level DL happens.

We would like to use ranking matrix formulate this action system:
\[ S = [s_{ij}]_{M \times N} \] stand for the state matrix of the system with \( i, j \in \{1,2,\ldots, M\} \) and \( i, j \in \{1,2,\ldots, N\} \)

**Proposition 1**
For agents/ jobs A={ai, i=1,…, MA} in the detection system, we have that,
\[ s_{ij} > 0 \] for all \( j \notin A \)
\[ s_{ij} = 0 \] for all \( j \in A \)

For 1<i<MA, then \( y_{ij} \geq 0 \) and \( s_{ij} > 0 \) (Figure 5).
\[ a_i \] defined as the Last agent of \( A \) and denoted by \( T(A) = a_i \), \( a_{kl} \) defined as the First agent where \( A \) and denoted by \( H(A) = a_{kl} \).

\[ z \] defined as the head resource of \( A \) and denoted by \( H(A) = z \). After \( P_i \) changing every state, a new matrix will be derivative: if \( i \neq k \) then \( \tilde{S} = [\tilde{s}_{ij}] \) as \( \tilde{s}_{ij} = s_{ij} \) or \( \tilde{s}_{ij} = s_{ij} - 1 \) while \( i = k \). Using set \( R = \{ r_j, j \in \{1,2,\ldots, M\} \} \) stand for M resources, using set \( P = \{p_i, i \in \{1,2,\ldots, N\} \} \) stand for N agents/jobs, all agents follows the predefined working procedure \( S = [s_{ij}]_{M \times N} \) stand for the system state at certain point. \( s \) stand for the position when agent/job \( p_i \) being processed on resource \( r_j \) (Figure 6.1).

If \( s_{ij} > 0 \) means \( r_j \) not being occupied by agent/job \( p_i \); yet, if \( s_{ij} = 0 \) then job \( p_i \) is occupying resource \( r_j \); if \( s_{ij} > 0 \) then job \( p_i \) finished tasks on \( r_j \). Hence, the ranking matrix of state changed from \( S_i \) to state \( S_i \) while agent/job \( p_i \) transferring position from \( r_j \) to \( r_s \).

**Proposition 2**
The second level deadlock exist under necessary and sufficient condition as \( GL \) where:

![Figure 4: Deadlock III level.]

![Figure 5: Simple system.]

![Figure 6.1: System with 2nd level deadlock.]
(a) \( k \in I \) as for \( \forall a \in I \), \( HR(a) = k \).
(b) \( \forall c \in I \) as \( HJ(b) = b \) there exist \( c_i \in I \) that \( s_{f_a} = 2 \), \( s_{r_c} = 0 \), and \( HJ(c) \neq b \) (Figure 6.2).

If apply Proposition 2 to the previous example in ranking matrix.

The state matrix will be:

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

Proposition 3

The high level deadlock exist under necessary and sufficient condition as \( L \subseteq 1 \) which:

(a) \( x,y \in J \) for

(i) for \( \forall a \in J \), \( HR(a) = x \), \( HR(a) = y \) = 0

(ii) \( \sum_{a \in J} [HR(a) - x] \cdot \sum_{a \in J} [HR(a) - y] \neq 0 \),

(b) \( \forall b \in I \) as \( HJ(b) = b \)

(i) \( k \in J \) as \( s_{k_b} = 2 \);

(ii) if \( c \in I \) as \( s_{c} = 0 \), then \( HJ(c) \neq b \), o.w., \( (k-x) \) \( (k-y) = 0 \), as \( d \in I \) and \( / \) by \( s_{d_k} = 3 \), \( s_{r_c} = 0 \), \( HR(d) \neq HR(b) \) (Figure 7).

If apply Proposition 3 to the previous example in ranking matrix.

The state matrix will be:

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

1. \( HR(1) = HR(2) = HR(4) = 3 \), \( HR(3) = HR(5) = 4 \) so condition (a) satisfied.
2. \( HR(2) = 2 \), \( HR(3) = 3 \), \( HR(4) = 4 \) and \( s_{c_2} = s_{c_3} = 2 \) so condition \( [b(i)] \) satisfied.
3. Resource 4 will be required by Agent 2 in two steps and available; also have \( s_{c_4} = 3 \), \( s_{f_3} = 0 \) and \( HR(2) = 3 \neq HR(3) = 4 \). Condition \( [b(ii)] \) satisfied as \( s_{c_4} = 2 \).

3.1 \( s_{c_2} = 2 \), \( s_{c_3} = 0 \), and \( HJ(3) = 3 \neq HJ(4) = 4 \).

3.2 \( s_{c_4} = 2 \), \( s_{c_3} = 0 \), and \( HJ(4) = 4 \neq HJ(1) = 2 \).

As mentioned above, we consider all collaborative action teams seeking to optimize global rewards, and we assume that we can use our reinforcement learning approach to model the corresponding multi-action stochastic systems and provide a search algorithm. Therefore, there is at least one action sequence that maximizes the expected return of all movements [20-26].

Definition 1

Set \( S \subseteq S \) be the system state of \( i \), where \( s_i = \{ A(\pi_1), A(\pi_2), \ldots A(\pi_n) \} \), \( \{A_i\} \) denote actions will be executed, \( \pi_i \) denotes the policies of action. The action \( A_i \) at state \( l \) and the reward value \( R(l, \pi) \), represent by:

\[
R(l, \pi) = \sum_{i=1}^{n} (J_i + M_i) \frac{C_{max}}{C_{max}}
\]

\( C_{max} \) denotes the makespan, \( \sum_{i=1}^{n} (J_i + M_i) \) represent the summary of jobs and machines been involved. Here \( P(\pi) \) represents the performance of policy \( \pi \), and \( R(\pi) \) represents the local action reward parameter.

Definition 2

Under the set \( S \subseteq S \), the policy:

\[
P(\pi) = E[R_i | s_0 \in S, \pi] = E[\sum_{i=1}^{n} (J_i + M_i) | \pi] = \lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} R_i
\]

is defined as the expected local reward \( R(\pi) \) and set discount factor \( \gamma \) to 1.

Evaluation

There are 40 classical scheduling benchmark problems for testing. The design of these problems adopts complex structure to increase difficulty. Additionally, if these systems are buffer-less, find scheduling will harder. Gantt chart can be drawn based on a DL-free timesheet obtained by each scheduling benchmark [27-31].

As shown in Figures 8 and 9, they present benchmark LA08 (15 × 5) and benchmark LA16 (10 × 10). We test the performance of our algorithm in this 40 benchmarks with backtracking counting’s, we also compare the running time between with and without DL detection. Algorithms is written in Matlab, and the workflow of our RL algorithm is shown in Figure 10.

The results of 40 benchmark testing with our algorithm are given in Table 1. From the results, we found that:
(1) All 40 benchmarks can be solved via algorithm within acceptable time frame.

(2) Our results are very close to the optimal solution. These shows our policy-based RL approach is effective in reducing time and cost.

(3) Due to the system difficulty, once the system becomes larger, the number of backtracking increases. Backtracking numbers are 0 for first 15 benchmark, and the number increases as the states increases. However, for all benchmark problems, our number of backtracking used is kept at a low level.

(4) Once a DL event occurs, our scheduling algorithm can rearrange and generate a new DL-free timesheet within 1 seconds. Therefore, we can assuming that our DL-free algorithm would be applied to other similar structure systems. Additionally, under more power computation system this algorithms making itself a qualified tool for real-time operation system (Table 1).

**Conclusion**

Based on the ranking matrix, graph model and reinforcement learning, a new corresponding DL detection algorithm is proposed by us, and using that the author analyzed the general pattern of high-level DL detection problem based on discrete system, using the classical reinforcement learning approach.

**Figure 8:** Optimal schedule of LA08.

**Figure 9:** Optimal schedule of LA16.

**Figure 10:** DL detection algorithm via RL.
forty benchmark problems. However due to the heavy computation, some work might took very long term, but this can be solved in time while the computation speed is exponential increasing.

This algorithm is developed under the buffer less environmental which is much more difficulty compare to real world. Therefore, it is worth believing that our algorithm should be extended to other resource sharing systems.

Based on this DL detection algorithms, relax some certain constrains new limited buffer DL detection algorithms can be developed and can be widely applied in the mechanical system, parallel computing system, and the future is quite bright.

References

### Table 1: Evaluation table.

<table>
<thead>
<tr>
<th>Problem (size)</th>
<th>2LD</th>
<th>3LD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Backtrack</td>
<td>Time</td>
</tr>
<tr>
<td>FT 06</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 01 (10 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 02 (10 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 03 (10 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 04 (10 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 05 (10 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 06 (15 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 07 (15 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 08 (15 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 09 (15 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 10 (15 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 11 (20 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 12 (20 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 0 13 (20 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 14 (20 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 15 (20 x 5)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 16 (10 x 10)</td>
<td>&lt;1 sec.</td>
<td>1</td>
</tr>
<tr>
<td>LA 17 (10 x 10)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 18 (10 x 10)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 19 (10 x 10)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 20 (10 x 10)</td>
<td>&lt;1 sec.</td>
<td>0</td>
</tr>
<tr>
<td>LA 21 (15x10)</td>
<td>&lt;1 sec.</td>
<td>21</td>
</tr>
<tr>
<td>LA 22 (15 x 10)</td>
<td>&lt;1 sec.</td>
<td>29</td>
</tr>
<tr>
<td>LA 23 (15 x 10)</td>
<td>&lt;1 sec.</td>
<td>143</td>
</tr>
<tr>
<td>LA 24 (15 x 10)</td>
<td>&lt;1 sec.</td>
<td>22</td>
</tr>
<tr>
<td>LA 25 (15 x 10)</td>
<td>&lt;1 sec.</td>
<td>108</td>
</tr>
<tr>
<td>LA 26 (20 x 10)</td>
<td>&lt;1 sec.</td>
<td>38</td>
</tr>
<tr>
<td>LA 27 (20 x 10)</td>
<td>&lt;1 sec.</td>
<td>36</td>
</tr>
<tr>
<td>LA 28 (20 x 10)</td>
<td>&lt;1 sec.</td>
<td>6</td>
</tr>
<tr>
<td>LA 29 (20 x 10)</td>
<td>&lt;1 sec.</td>
<td>196</td>
</tr>
<tr>
<td>LA 30 (20 x 10)</td>
<td>&lt;1 sec.</td>
<td>23</td>
</tr>
<tr>
<td>LA 31 (30 x 10)</td>
<td>&lt;1 sec.</td>
<td>33</td>
</tr>
<tr>
<td>LA 32 (30 x 10)</td>
<td>&lt;1 sec.</td>
<td>73</td>
</tr>
<tr>
<td>LA 33 (30 x 10)</td>
<td>&lt;1 sec.</td>
<td>71</td>
</tr>
<tr>
<td>LA 34 (30 x 10)</td>
<td>&lt;1 sec.</td>
<td>94</td>
</tr>
<tr>
<td>LA 35 (30 x 10)</td>
<td>&lt;1 sec.</td>
<td>68</td>
</tr>
<tr>
<td>LA 36 (15 x 15)</td>
<td>&lt;1 sec.</td>
<td>69</td>
</tr>
<tr>
<td>LA 37 (15 x 15)</td>
<td>&lt;1 sec.</td>
<td>239</td>
</tr>
<tr>
<td>LA 38 (15 x 15)</td>
<td>&lt;1 sec.</td>
<td>339</td>
</tr>
<tr>
<td>LA 39 (15 x 15)</td>
<td>&lt;1 sec.</td>
<td>35</td>
</tr>
<tr>
<td>LA 40 (15 x 15)</td>
<td>&lt;1 sec.</td>
<td>415</td>
</tr>
</tbody>
</table>

ISSN: 2169-0316