AN ASYNCHRONOUS REINFORCEMENT LEARNING HYPER-HEURISTIC ALGORITHM FOR FLOW SHOP PROBLEM

Wen Shi, Xueyan Song, Cuiling Yu, Jizhou Sun
School of Computer Science and Technology, Tianjin University
Tianjin, China
shiwen@tju.edu.cn, jzsun@tju.edu.cn, yucuiling1989@163.com, songxy@tju.edu.cn

ABSTRACT
This paper investigates a hyper-heuristic algorithm for the permutation flow shop problem (FSP) to find a sequence to minimize the makespan. In comparison with existing approaches, our proposed hyper-heuristic algorithm based on multi-agent architecture includes two levels: low level heuristic agents (LLHAs) do local search in the solution domain and hyper-heuristic agent (HHA) manages low level heuristic agents with reinforcement learning. The LLHAs improve the current solution by local search and send it to the HHA. Depending on the last few performances of LLHAs, the HHA decide whether or not to accept the current received solution as an initial solution for the next local search. Simulation studies demonstrate that the hyper-heuristic with asynchronous parallel reinforcement learning yields better solutions than other algorithms.

KEY WORDS
Agent-based and Multiagent Systems, Machine Learning, Permutation Flow Shop Scheduling, Hyper-heuristic

1. Introduction

With a strong engineering background, FSP is a strongly NP-complete combinatorial optimization problem[1]. The object of FSP is to schedule n-jobs on m-machines to minimize the makespan which is the time from the first job starting on the first machine to the last job finish on the last machine.

The earliest optimal solution for FPS can be dated back to 1954[2]. There are two kinds of algorithms for FPS: constructive methods and improvement methods. Constructive algorithms start with an empty solution and gradually build a complete solution. An early constructive algorithm is proposed by Campbell et al.[3] which is called CDS. CDS method clusters all the machines into two virtual machines and solves the generated two machine problem repeatedly. Another famous constructive heuristic is NEH algorithm which is proposed by Nawaz et al.[4]. NEH algorithm sorts all the jobs by non-increasing the total processing time on all the machines, select the first two jobs and make a partial sequence. Then the other jobs are inserted into the current sequence one by one which minimises the makespan.

Suliman[5] developed an improvement heuristic with a job chair exchange mechanism with a directionality constraint. In addition to these domain specific algorithms, some popular artificial intelligence techniques like genetic algorithms[6], artificial neural networks[7] and simulated annealing[8, 9] are used to solve FSP. However, according to the "no-free lunch" theorem, no heuristic search algorithm is suitable for all optimization problems or all instances in one problem. Building a general improvement heuristic for FSP to choose a suited heuristic in each determination point during the search is one of the motivations of hyper-heuristics in this paper.

A survey of hyper-heuristic was described by Burke et al.[10]. The idea behind of hyper-heuristics can be dated back to the early 1960s[11]. In early 2000s, hyper-heuristics was introduced to describe the idea of "heuristics to choose heuristics" in the context of combinatorial optimization[12]. The early hyper-heuristic is to manage constructive heuristics to build a complete solution gradually. Then, improvement hyper-heuristics are proposed to select and apply low level heuristics to improve the current solution. Further studies in hyper-heuristics try to automatically generate new heuristics suited to a given domain from a set of potential heuristic components[13].

The main process in hyper-heuristics includes heuristic selection and move acceptance. In reference[10], different kinds of heuristic selection mechanisms such as Simple Random (SR), Random Permutation (RP) and Greedy (GR)[12, 14]. The process of heuristic selection can also make use of the reinforcement learning[15, 16] to improve the decision making process. The reinforcement learning mechanism is shown in Figure 1. In this kind of strategy, weights represent the performance of heuristics and could be treated as a guide of the search process. Nareyek[17] makes uses of reinforcement learning as a low level heuristic selection method and set a weight for each low level heuristic. Burke et al.[18] and Özcan et al.[19] combined reinforcement learning with tabu search and great deluge. However, in these heuristic selection methods, heuristics are arranged one by one but not in parallel. Obviously, the efficiency will decrease when only one heuristic does a local search at any time. For distributing the hyper-heuristic having parallel execution of several low level heuristics, Ouelhadj and Petrovic[20] proposed an agent-based cooperative hyper-heuristic framework. The agent-based heuristics could not only...
cooperate but also compensate for weakness each other. Results in this study demonstrated that the cooperative hyper-heuristic in parallel outperformed the sequential ones. In reference [20], synchronous and asynchronous cooperative mechanisms are in combination with existing move acceptance strategies such as Greedy Improving Only[21], Greedy All Moves[21], Tabu Search[22], Simulated Annealing[21] et al. but without learning.

In this paper, a hyper-heuristic base on multi-agent framework for FSP is proposed. We embed reinforcement learning into a hyper-heuristics multi-agent framework. And we propose a deterministic move acceptance with non-stationary reinforcement learning described in reference [17]. The HHA manages LLHAs with reinforcement learning by generating online weights for each LLHAs based on their performances and decides whether accept the solution as the initial solution for the next iteration according to weights of LLHAs. HHA cooperates with LLHAs asynchronously. The LLHAs contains five exciting improvement heuristics: Suliman[5], SWAP[6], inversion[6], insertion[6] and permutation[10]. Simulation studies demonstrate that the asynchronous parallel cooperative hyper-heuristic with reinforcement learning outperforms the ones with deterministic or non-deterministic move acceptance and other flow-shop algorithms.

![Figure 1. The reinforcement learning mechanism](image)

### 2. Problem description

The goal of the FSP is to find an optimal schedule for $n$ jobs $(i = 1...n)$ on $m$ machines $(j = 1...m)$. A job consists in $m$ operations and the $j$ operation of each job must be processed on machine $j$. If the job completes on machine $j-1$ meanwhile the machine $j$ is free, it can start on machine $j$. Meanwhile, one machine could process one job and one job could only operate on one machine at a time. Each operation has a known processing time $p_{i,j}$.

The competition time $C_{i,j}$ for the job $i$ on the machine $j$ can be calculated using the following formulate:

$$C_{i,1} = p_{i,1}$$

$$C_{i,1} = C_{i-1,1} + p_{i,1} \quad i = 2,...,n$$

$$C_{1,j} = C_{1,j-1} + p_{1,j} \quad j = 2,...,m$$

$$C_{i,j} = \max\{C_{i-1,1}, C_{i,j-1}\} + p_{i,j} \quad i = 2,...,n \quad j = 2,...,m$$

And the value of objective function $C_{\text{max}}$ is the completion time for the last operation on the last machine. The objective is the minimisation of makespan $C_{\text{max}}$.

### 3. Hyper-heuristic for FPS

#### 3.1 Hyper-heuristic Agent

The HHA proposed in this paper makes use of heuristic selection methods with learning and deterministic move acceptance. Meanwhile, as an agent, the HHA could cooperate with LLHAs asynchronously.

First of all, the HHA generates an initial solution and broadcast to all LLHAs. The LLHAs receive the solution, improve the solution by local search and send it back to the HHA. The HHA has a list to store the weights of the LLHAs which depended on the performance of LLHAs before. When it received a solution from one LLHA, HHA adjusts the weight of the LLHA based on the received solution and decides the initial solution for the next iteration.

The pseudo code of process of HHA is:

```plaintext
Set weight of LLHA as $w_i$
Send the initial solution to LLHA;
While iteration < maximum number of iterations
    Do
        While receive the current solution from LLHA
            Do
                If the current solution is better than the local best solution then
                    Update the local best solution as the current solution;
                    If the current solution is better than the global best solution then
                        Update the global best solution as the current solution;
                        End if
                Else
                    $w_k = \text{award}(w_k)$;
                    $w_k = \text{punish}(w_k)$;
                    If $w_k > w_j$, $\forall j \in \{1..n\} \cap j \neq k$ then
                        Update the local best solution as the current solution;
                    End if
                End if
            End if
        End while
    End while
Set weight of LLHA as $w_i$
Initialize initial solution, local best solution and global best solution;
iteration = 0;
For $i = 1$ to $n$
    Set weight of LLHA as $w_i$;
    Send the initial solution to LLHA;
    While iteration < maximum number of iterations
        Do
            While receive the current solution from LLHA
                Do
                    If the current solution is better than the local best solution then
                        Update the local best solution as the current solution;
                        If the current solution is better than the global best solution then
                            Update the global best solution as the current solution;
                            End if
                    Else
                        $w_k = \text{award}(w_k)$;
                        $w_k = \text{punish}(w_k)$;
                        If $w_k > w_j$, $\forall j \in \{1..n\} \cap j \neq k$ then
                            Update the local best solution as the current solution;
                        End if
                    End if
                End if
            End while
        End while
    End while
```

The initial solution is set as the local best solution; iteration++;

176
Send the initial solution to $LLHA_k$;
End do
End do
Terminate;

3.2 Low Level Heuristic Agents

The LLHAs in hyper-heuristic framework are in charge of local search in the solution domain. The LLHAs cooperate with HHA in parallel as described in Figure 2.

In this paper, two kinds of heuristics are introduced. One is mutational heuristics which make a random perturbation and do not expected to produce a better solution after they are applied. Mutational heuristics in our hyper-heuristic for FPS include:

- Insertion heuristic: Select a job randomly and move it to another position in the sequence of scheduled jobs;
- Inversion heuristic: Select a sub-sequence of jobs randomly and reverses the positions of these jobs;
- Swap heuristic: Select two jobs in the sequence of scheduled job randomly and exchange their positions;
- Permutation heuristic: Select a sub-sequence of jobs and do a random sequence permutation;

The other kind of heuristics is called hill climber. The aim of hill climber is to improve the candidate solution. The suliman heuristic proposed by Suliman[5] is introduced as hill climber in our hyper-heuristic framework. The suliman heuristic improves the solution with a job pair exchange with a directionality constraint.

The pseudo code of process of LLHAs is:

Do
    If receive the initial solution from HHA then
        local_iteration = 0;
        While local_iteration < maximum number of local iterations
            Do
                Find the current solution by local search strategy;
                local_iteration++;
            End do
        End if
    Send the current solution back to HHA;
End do

4. Performance evaluation

4.1 Simulation Environment

We implemented the hyper-heuristic based on multi-agent architecture using Java Agent Development framework (Jade)[23], which is an open source for peer-to-peer agent based applications, in the hardware/software platform:
Intel Pentium Dual E2160, 1.80GHz, 1.79GHz, 2-GB random access memory, Microsoft Windows XP OS.

We consider 8 different sizes of the flow shop secluding instances given by Taillard\[24\] including 20\,*\,*5, 20\,*\,*10, 20\,*\,*20, 50\,*\,*5, 50\,*\,*10, 50\,*\,*20, 100\,*\,*5 and 100\,*\,*10. We select one instance of each size. Similar to reference \[20\], we also calculate the average percentage makespan increase over the lower bound (LB) of $C_{\text{MAX}}$ to evaluate different approach for FSP. Thus, smaller values in Figure 4 are better results. The weight updating mechanisms we used are $\omega_a \leftarrow \omega_a + 1$ and $\omega_a \leftarrow \omega_a - 1$ as described in reference \[17\]. The termination condition is a maximum number of 20000 iterations. To compare performance of different algorithms fairly, we make use of a large enough number of iterations. In Figure 3, performances of algorithms for different number of iterations in one case of FSP are shown. After about 5000 iteration, local best solutions of every algorithm tend to be stable.

To evaluate the performance of our proposed algorithm, firstly we implement some move acceptance strategies combined with parallel heuristic selection for comparison:

- Deterministic move acceptance strategies: tabu search (TS) and simulated annealing (SA).
- Non-deterministic move acceptance strategies: greedy improving only (IO) and greedy all moves (AM).

4.2 Experimental results

In Figure 4, the results of different algorithms are shown. Hyper-heuristic with asynchronous parallel reinforcement learning (abbreviated as ARL in Figure 4) outperforms others with deterministic or non-deterministic move acceptance strategies. For some small scale cases such as 20\,*\,*5, 50\,*\,*5 and 100\,*\,*5, the differences between the results are not obvious. But for large scale cases, our proposed algorithm yields better solutions apparently.

Table 1 Comparison with Suliman heuristic algorithm

<table>
<thead>
<tr>
<th></th>
<th>20,*,*5</th>
<th>20,*,*10</th>
<th>20,*,*20</th>
<th>50,*,*5</th>
<th>50,*,*10</th>
<th>50,*,*20</th>
<th>100,*,*5</th>
<th>100,*,*10</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD+ARL</td>
<td>5.891</td>
<td>12.155</td>
<td>23.48</td>
<td>0.516</td>
<td>9.529</td>
<td>16.006</td>
<td>1.839</td>
<td>5.939</td>
</tr>
<tr>
<td>NEH+ARL</td>
<td>5.62</td>
<td>11.609</td>
<td>23.061</td>
<td>0.442</td>
<td>9.46</td>
<td>15.934</td>
<td>1.416</td>
<td>1.632</td>
</tr>
<tr>
<td>NEH+Suliman</td>
<td>5.814</td>
<td>12.431</td>
<td>25.118</td>
<td>0.627</td>
<td>9.77</td>
<td>16.006</td>
<td>1.416</td>
<td>1.667</td>
</tr>
</tbody>
</table>

As an improvement algorithm for FSP, our proposed algorithm also outperformed other existing improvement algorithm such as Suliman heuristic (SC) as described in Table 1. The headings of the table are 8 cases of FSP. The results in Table 1 are also the average percentages makespan increase over LB. For testing the performance, we combine these two improvement algorithms to two constructive algorithms including random (RD) and NEH respectively. Results show that hyper-heuristic with asynchronous reinforcement learning performs better than SC whether collaborates with RD or NEH. Meanwhile, our method is so effective that the results of RD-ARL are
near to those of NEH-ARL even if RD is worse than NEH evidently.

5. Conclusion

Hyper-heuristics solves the search problem by comprising different kinds of heuristic methods with the collective goal. In order to further improve the search efficiency of hyper-heuristic with sequential reinforcement learning mechanism, in this paper, we propose parallel reinforcement learning with a hyper-heuristics multi-agent framework and apply them FSP. Simulation results show that the hyper-heuristic with asynchronous parallel reinforcement learning mechanism yield better solutions than those with deterministic or non-deterministic move acceptance strategies. And it is also an effective improvement algorithm for FSP.

In our future work, multi-agent reinforcement learning will be applied to the hyper-heuristics. And more widely comparative work on hyper-heuristics multi-agent competition would be carried out.

Acknowledgements

This study is supported by grants from National Natural Science Foundation of China and Civil Aviation University of China(61039001) and Tianjin Municipal Science and Technology Commission(11ZCKFGX04200).

References