


ORIGINAL ARTICLE

Open Access



# Necessary and Sufficient Conditions for Feasible Neighbourhood Solutions in the Local Search of the Job-Shop Scheduling Problem

Lin Gui<sup>1</sup>, Xinyu Li<sup>1\*</sup> , Liang Gao<sup>1</sup> and Cuiyu Wang<sup>1</sup>

## Abstract

The meta-heuristic algorithm with local search is an excellent choice for the job-shop scheduling problem (JSP). However, due to the unique nature of the JSP, local search may generate infeasible neighbourhood solutions. In the existing literature, although some domain knowledge of the JSP can be used to avoid infeasible solutions, the constraint conditions in this domain knowledge are sufficient but not necessary. It may lose many feasible solutions and make the local search inadequate. By analysing the causes of infeasible neighbourhood solutions, this paper further explores the domain knowledge contained in the JSP and proposes the sufficient and necessary constraint conditions to find all feasible neighbourhood solutions, allowing the local search to be carried out thoroughly. With the proposed conditions, a new neighbourhood structure is designed in this paper. Then, a fast calculation method for all feasible neighbourhood solutions is provided, significantly reducing the calculation time compared with ordinary methods. A set of standard benchmark instances is used to evaluate the performance of the proposed neighbourhood structure and calculation method. The experimental results show that the calculation method is effective, and the new neighbourhood structure has more reliability and superiority than the other famous and influential neighbourhood structures, where 90% of the results are the best compared with three other well-known neighbourhood structures. Finally, the result from a tabu search algorithm with the new neighbourhood structure is compared with the current best results, demonstrating the superiority of the proposed neighbourhood structure.

**Keywords** Scheduling, Job-shop scheduling, Local search, Neighbourhood structure, Domain knowledge

## 1 Introduction

The job-shop scheduling problem (JSP), one kind of classic scheduling problem [1], is widespread in the modern manufacturing system [2]. It can be described as follows: there are  $n$  jobs, and each has  $m$  operations needing to be processed on  $m$  different machines. The order

of machines used for operations in each job could be different. Each machine can only process one operation at one time, and one operation can only be processed on one machine at one time [3]. Processing cannot be interrupted. The goal of this problem is to obtain the solution with minimum makespan by arranging the processing sequence of jobs for each machine [4].

The JSP is a proven NP-Complete problem [5], for which meta-heuristic algorithms are more popular than other methods. The key to ensuring a high-quality solution in meta-heuristic algorithms is to use a good local search strategy [6], and the existing literature shows that the algorithm with this strategy [7–10], at least as

\*Correspondence:

Xinyu Li  
lixinyu@mail.hust.edu.cn

<sup>1</sup> State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

a part of the algorithm [11–14], could be easier to get a better result. The basic process of a local search strategy is to obtain a set of neighbourhood solutions from the current solution through the neighbourhood structure and then select one as the current solution for the next iteration. In 1996, a comprehensive survey [15] summarized six effective neighbourhood structures named N1–N6. In 2007, Zhang et al. [9] proposed a new neighbourhood structure based on N6 [16], which enlarged the search space of the neighbourhood and was called N7. In 2022, Xie et al. [17] obtained more neighbourhood solutions by moving the operations in the critical path outside the critical path. These neighbourhood structures are widely used in many algorithms and achieved tremendous success [18]. Of course, there are many other neighbourhood structures, but most of them are based on the above seven neighbourhood structures. In fact, two issues have been driving the design of neighbourhood structures in the JSP: (1) how to ensure the neighbourhood solution is feasible; (2) how to increase (reduce) the neighbourhood solutions with (without) improvement on the current solution. They can improve the search results of the algorithm by ensuring the feasibility and effectiveness of the neighbourhood solutions.

In this paper, we mainly study the feasibility of neighbourhood solutions in the JSP. By analyzing the causes of infeasible solutions, we propose sufficient and necessary constraint conditions for feasible neighbourhood solutions and the properties of these constraints are proved. On this basis, a new neighbourhood structure is designed to ensure that the local search is carried out thoroughly, and a fast computational method is proposed to cope with the complexity of this neighbourhood structure. Obviously, these conditions not only ensure the feasibility of neighbourhood solutions but also improve the effectiveness of local searches. Experiments at different levels were carried out to demonstrate the reliability and superiority of the new neighbourhood structure. The main purpose of this paper is not to design a meta-heuristic algorithm for the JSP, but to mine the domain knowledge to ensure the feasibility of neighbourhood solutions. It is hoped that by mining the domain knowledge, we can have a deeper understanding of the JSP and provide theoretical support for the design of algorithms for it.

The remainder of the paper is organized as follows. Section 2 uses the disjunctive graph model to describe the JSP. The domain knowledge in the local search of the JSP and the related applications are presented in Section 3. The necessary and sufficient conditions for the feasible neighbourhood solution and the fast calculation method proposed are provided in Section 4. The algorithm used in this paper and the experimental results are

presented in Sections 5 and 6, respectively. The summary and the prospect are carried on in Section 7.

### 2 Description of JSP

The JSP is a complex combinatorial optimization problem that needs to arrange the processing sequence of the jobs for each machine to optimize some objective functions. In this paper, the objective function is to minimize the total completion time. The disjunctive graph model is usually used to describe the JSP. A disjunctive graph can be represented by a triple  $G=(V, A, E)$ .  $V$  is the set of vertices in  $G$ , where each vertex represents an operation of a job. It is worth noting that there exist two virtual operations,  $s$  and  $e$ , which are the starting and ending points of all jobs, respectively.  $A$  is the set of conjunctive arcs in  $G$ , where each arc represents the processing sequence of different operations in the same job and is shown by solid lines with an arrow.  $E$  is the set of disjunctive arcs in  $G$ , where each arc connects operations processed on the same machine and is shown by a dashed line. Different from the existing literature, this paper sets the weight of each vertex in the disjunctive graph, instead of the arc length, as the processing time of the corresponding operation, and the weight of the two virtual operations  $s$  and  $e$  is 0. Therefore, it is easy to know that for the JSP, we need to determine a unique direction for each disjunctive arc in the disjunctive graph, which is to determine the processing sequence of each operation on each machine. In the new disjunctive graph, the maximum weight sum of the path from  $s$  to  $e$  is called the critical path, and the weighted sum of the path is the makespan of the scheduling.

A simple example is used to explain the symbols and concepts further. Assume that there exist three jobs, which have three operations each  $J1=(O_{1,1}, O_{1,2}, O_{1,3})$ ,  $J2=(O_{2,1}, O_{2,2}, O_{2,3})$ ,  $J3=(O_{3,1}, O_{3,2}, O_{3,3})$ . Here,  $O_{j,i}$  means the  $i$ th operation of the  $j$ th job. These operations are processed on three machines  $M1:(O_{1,1}, O_{2,1}, O_{3,1})$ ,  $M2:(O_{1,2}, O_{2,3}, O_{3,3})$ ,  $M3:(O_{1,3}, O_{2,2}, O_{3,2})$ . The disjunctive graph of this example can be shown in Figure 1.

The information of each node or operation can be represented by  $[m_{j,p} p_{j,i}]$ , where  $m_{j,i}$  and  $p_{j,i}$  means the

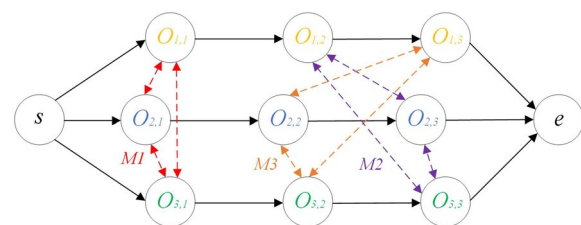


Figure 1 The disjunctive graph for JSP

processing machine and the processing time of  $O_{j,i}$ . Assume that the information of the operations is  $J1:([1,5], [2,2], [3,4])$ ,  $J2:([1,3], [3,2], [2,4])$ ,  $J3:([1,3], [3,2], [2,2])$ . When the direction of the disjunctive arc in Figure 1 is determined, we can obtain a scheduling scheme, and it can be shown in Figure 2 (Some of the edges, like the edge from  $O_{2,1}$  to  $O_{1,1}$ , the edge from  $O_{2,2}$  to  $O_{1,3}$  and the edge from  $O_{3,3}$  to  $O_{1,2}$ , are omitted to make the graph clearer).

Though the disjunctive graph can represent the processing sequence of different operations on the same machine, the exact start time and finish time of each operation cannot be obtained. The Gantt chart is usually used to represent the final result of the scheduling problem, and the Gantt chart of the feasible solution of the JSP is shown in Figure 3. The critical path of the solution in the disjunctive graph can be easily found in the Gantt chart, and it is connected and marked by solid red lines. The block formed by operations processed on the same machine in the critical path and no-wait between any two adjacent operations can be called the critical block, as shown in Figure 3, where each of them is marked by a black dotted box.

However, if the direction of the disjunctive arc is arbitrarily determined, infeasible solutions may appear in the JSP. For example, if we exchange the processing order of  $O_{3,2}$  and  $O_{1,3}$  in Figure 2, the obtained disjunctive graph of the solution is shown in Figure 4 (Some of the edges, like the edge from  $O_{2,1}$  to  $O_{1,1}$ , the edge from  $O_{2,2}$  to  $O_{3,2}$

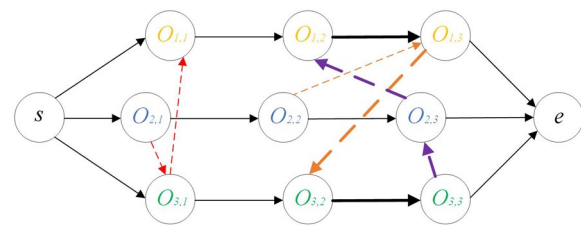


Figure 4 The disjunctive graph of an infeasible solution

and the edge from  $O_{3,3}$  to  $O_{1,2}$ , are omitted to make the graph clearer). In this disjunctive graph, the operations  $O_{1,2}$ ,  $O_{1,3}$ ,  $O_{3,2}$ ,  $O_{3,3}$  and  $O_{2,3}$  form a closed loop, and then we do not know which operation should process first. That is why a solution will be infeasible if a closed-loop exists in the disjunctive graph. Therefore, in the disjunctive graph model of the JSP, its decision variable is the direction of each disjunctive arc, and its objective function is to ensure the feasibility of the solution and minimize the weighted sum of the critical path of the feasible solution.

### 3 Domain Knowledge in Local Search of JSP

Domain knowledge is a general term, which means the knowledge of a specialized discipline or field [19]. It has different meanings in different contexts. In this paper, domain knowledge refers to the properties of the optimization problem itself that can be used to design efficient optimization algorithms. When the domain knowledge of a problem has been fully explored, we can have a more profound understanding of the problem itself and design the strategies or methods beneficial to problem-solving. Although the domain knowledge is very important, few scholars discussed the domain knowledge of the problem for the design of neighbourhood structure in the existing literature. Instead of it, most scholars have designed a neighbourhood structure and then proved its effectiveness by experiments. Although the results obtained by these two methods are similar, the first way can help

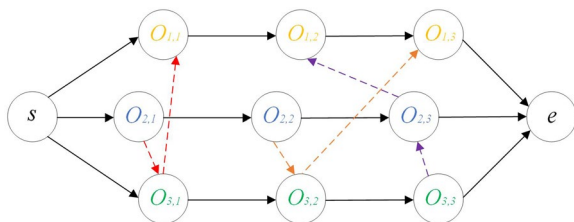


Figure 2 The disjunctive graph of a feasible solution

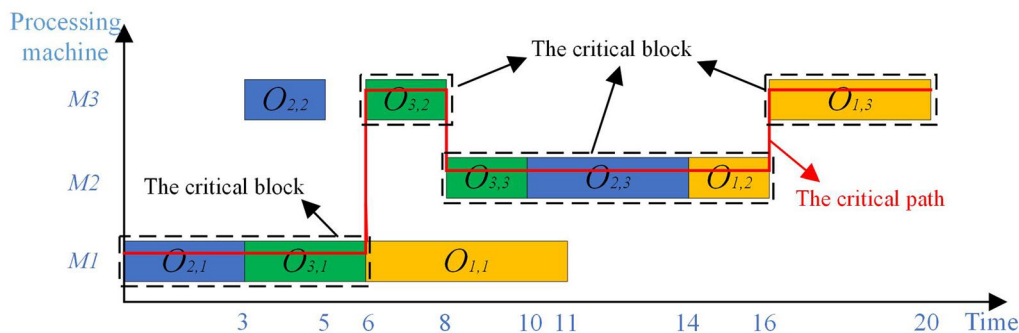


Figure 3 The Gantt chart of the feasible solution

us have a deeper understanding of the problem, so as to guide the future research direction.

In this section, we summarize the domain knowledge used to design neighbourhood structures in the JSP and classify it into two categories: (1) the domain knowledge to guarantee the effectiveness of neighbourhood structures; (2) the domain knowledge to ensure that neighbourhood solutions are feasible. Based on the domain knowledge, we introduce the existing well-known neighbourhood structures and show which relevant domain knowledge is applied to these neighbourhood structures. Finally, we point out the shortcomings of the existing domain knowledge and provide future research directions. To better describe this domain knowledge, it is necessary to understand the following definitions of symbols: the critical path is a path with the maximum weight sum from  $s$  to  $e$  in the disjunctive graph of a solution; the critical block is the block formed by operations processed on the same machine in the critical path and no wait between any two adjacent operations;  $u, v$  are two operations in the same critical block;  $O_{j,i}$  is the  $i$ th operation in the  $j$ th job;  $m_{O_{j,i}}$  is the machine used to process  $O_{j,i}$ ;  $p_{O_{j,i}}$  is the processing time of the  $O_{j,i}$ ;  $jp[O_{j,i}]$  is the operation in the same job as  $O_{j,i}$ , and processing just before  $O_{j,i}$ ;  $js[O_{j,i}]$  is the operation in the same job as  $O_{j,i}$ , and processing just after  $O_{j,i}$ ;  $mp[O_{j,i}]$  is the operation processed on the same machine as  $O_{j,i}$ , and processing just before  $O_{j,i}$ ;  $ms[O_{j,i}]$  is the operation processed on the same machine as  $O_{j,i}$ , and processing just after  $O_{j,i}$ ;  $F(O_{j,i})$  is the maximum weighted sum of a path from  $s$  to  $O_{j,i}$  in the disjunctive graph of the solution ( $O_{j,i}$  is not included);  $R(O_{j,i})$  is the maximum weighted sum of a path from  $O_{j,i}$  to  $e$  in the disjunctive graph of the solution ( $O_{j,i}$  is not included).

### 3.1 Domain Knowledge to Guarantee the Effectiveness and the Feasibility of Neighbourhood Structures

In this part, we will introduce some propositions, among which the first three propositions are to improve the effectiveness of local search, and the last three propositions are to ensure that the neighbourhood solutions obtained by the local search are feasible. Of course, there are some other properties used to improve the effectiveness and feasibility of local search in the existing literature, but most of them are expanded or improved on these basic propositions, which are not mentioned here. This paper only briefly introduces these propositions, and please refer to relevant references for specific proofs.

**Proposition 1.** *When the order of the operations on the critical path does not change, the makespan will not decrease no matter how the order of other operations changes [20].*

**Proposition 2.** *If the change of operations order occurs in a critical block, the makespan will not decrease when neither the first nor the last operation of the critical block does not change [21].*

**Proposition 3.** *When only the first (last) operation of the first (last) critical block in the critical path changes, the makespan will not decrease [8].*

Proposition 1 was proposed by Potts for the single-machine scheduling problem in 1980 and was subsequently used for branch and bound. Although it was not proposed for the JSP, the domain knowledge contained in it is also applicable to the JSP [22, 23]. The critical path is equivalent to the processing bottleneck. When the processing bottleneck does not change, the makespan will not decrease no matter how the order of other operations changes. The proposal and application of Proposition 1 significantly reduce the invalid search in local searches. Propositions 2 and 3 further reduce the scope of local search based on Proposition 1 so that the efficiency of local search can be further improved.

**Proposition 4.** *The neighbourhood solution generated by changing the position of any two adjacent operations on any critical block cannot be infeasible [24].*

**Proposition 5.** *Two operations,  $u$  and  $v$ , are processed on the same machine, and  $u$  processes before  $v$ . When it is satisfied  $F(u) + p_u \geq F(jp[v]) + p_{jp[v]}$ , the neighbourhood solution generated by changing the position of  $v$  to the position just before  $u$  must be feasible [16].*

**Proposition 6.** *Two operations,  $u$  and  $v$ , are processed on the same machine, and  $u$  processes before  $v$ . When it is satisfied  $R(v) + p_v \geq R(js[u]) + p_{js[u]}$ , the neighbourhood solution generated by changing the position of  $u$  to the position just after  $v$  must be feasible [16].*

The above three propositions are domain knowledge to ensure that the neighbourhood solutions of local search are feasible. Proposition 4 is also proposed based on Proposition 1. Through Proposition 4, we can exchange any two adjacent operations on the critical block without generating infeasible solutions. Propositions 5 and 6 illustrate the constraints to ensure the neighbourhood solutions are feasible when one operation is inserted forward and backwards, respectively. With these two constraints, local search is no longer limited to the exchange of adjacent operations. However, these two constraints are not sufficient and necessary conditions to ensure that a neighbourhood solution is feasible, which will miss part of the feasible



neighbourhood solution in the local search, resulting in inadequate local search.

### 3.2 Neighbourhood Structures Designed by the Domain Knowledge in the JSP

The neighbourhood structure is a rule to generate a slight disturbance in a given feasible solution to obtain a set of neighbourhood solutions for local search. To make the neighbourhood structures in JSP more efficient, they generally follow two rules below [25]: (a) disturbances must occur in the critical path; (b) the neighbourhood solutions obtained from the current solution must be feasible ones. Obviously, these two rules correspond to two types of domain knowledge used for local search in the JSP: effectiveness and feasibility. Based on the domain knowledge in Section 3.1, researchers designed various efficient neighbourhood structures. The emergence of these neighbourhood structures has greatly improved the solving quality of the JSP and has been widely cited by many scholars.

*N1*: A neighbourhood solution can be obtained by exchanging the processing order of any two adjacent operations in the critical block of the current solution [24], as shown in Figure 5. *N1* is proposed based on the domain knowledge in Propositions 1 and 4. This neighbourhood structure can obtain many neighbourhood solutions, but most are not better than the current one. Because of the easy implementation, it is usually used as a kind of slight disturbance inserted into other neighbourhood structures to form a variable neighbourhood structure.

*N5*: A neighbourhood solution can be obtained by exchanging the first or last two operations in one critical block of the current solution. However, the solutions obtained by exchanging the first two operations in the first critical block and exchanging the last two operations in the last critical block are left out [8]. The movement of this neighbourhood structure is shown in Figure 6. *N5* uses the domain knowledge in Propositions 1 and 2, 3 and 4. *N5* applies the domain knowledge in Propositions 2 and 3 based on *N1*, which makes the local search with *N5* much more efficient than *N1*.

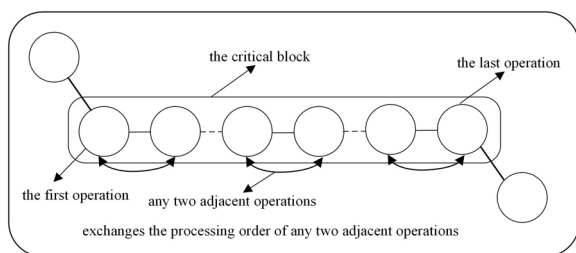


Figure 5 The *N1* neighbourhood structure

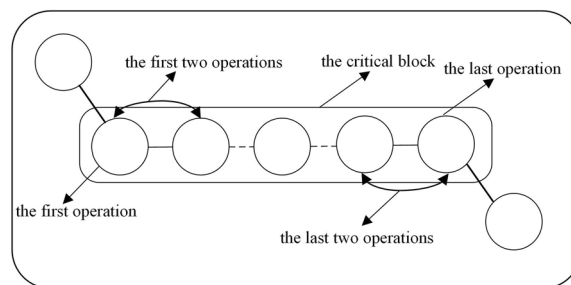


Figure 6 The *N5* neighbourhood structure

*N6*: A neighbourhood solution can be obtained by moving any operation in the critical block to the position just before (after) the first (last) operation under the conditions described in Propositions 5 and 6. *N6* is a deeper understanding of Proposition 2, that is, the first (last) operation on the critical block can be changed by inserting the operation inside the critical block into the position just before (after) the first (last) operation, rather than just by the exchange of adjacent operations [16], as shown in Figure 7. At the same time, infeasible solutions are directly removed by using the properties of Propositions 5 and 6.

*N7*: *N7* is a deeper application of Proposition 2 based on *N6*. In addition to the neighbourhood solutions obtained by *N6*, a neighbourhood solution can also be obtained by moving the first (last) operation of a critical block to the position just after (before) other operations inside this critical block under the conditions described in Propositions 5 and 6 [9]. The movement of *N7* can be shown in Figure 8. *N7* can obtain more effective domain solutions than *N6*, so it can conduct a more adequate local search.

From the above description, we can see that although there is not much domain knowledge for local search in the JSP, different scholars can design different application methods from the same proposition. For example, for Proposition 2, Balas et al. chose to insert the first or last operation into the critical block, but Zhang et al. inserted

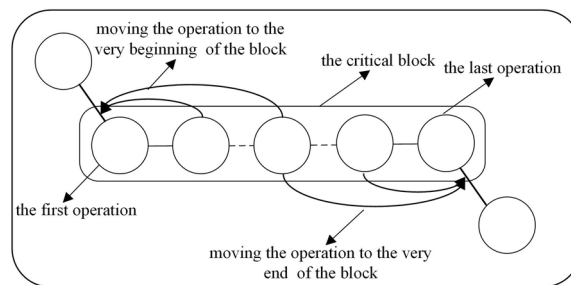
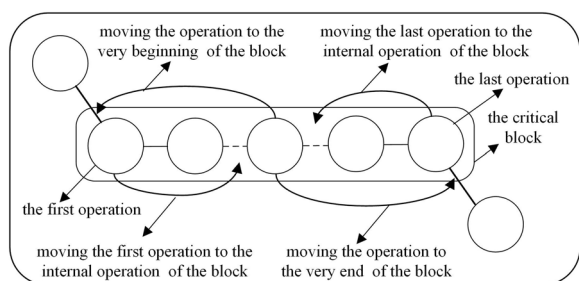


Figure 7 The *N6* neighbourhood structure



**Figure 8** The N7 neighbourhood structure

the operations in the critical block to the position just before (after) the first (last) operation on this basis. For Proposition 1, most scholars changed the sequence of the operations inside the critical block, but Xie et al. [26] proposed to move the operations inside the critical path to the outside of the critical path, which also changed the processing sequence on the critical path. There are also many other types of neighbourhood structures in the JSP, such as variable neighbourhood structure [27], multi-operation joint movement neighbourhood structure [28], but the domain knowledge used in these neighbourhood structures is similar, so they are not described here.

### 3.3 Summary of the Domain Knowledge for Local Search in the JSP

This section describes the domain knowledge for local search in the JSP and shows how to use this domain knowledge to design effective neighbourhood structures. More detailed descriptions can be obtained in related literature. These neighbourhood structures play an essential role in solving the JSP, and their achievements have been widely applied to different algorithms, and many benchmarks have been successfully refreshed.

Although effective neighbourhood structures can be designed based on the existing domain knowledge, it needs to be further explored in JSP due to the shortcomings of the existing domain knowledge. It can also be discussed in terms of effectiveness and feasibility. (1) The effectiveness: with the existing neighbourhood structures, a large number of neighbourhood solutions without improvement on the current solution can still be obtained. Therefore, domain knowledge that can further distinguish whether there is an improvement on the current solution is needed to improve the efficiency of local search. (2) The feasibility: although the constraint conditions to ensure the feasibility of neighbourhood solutions in existing literature are helpful, the disadvantage of them is that they may regard some feasible neighbourhood solutions as infeasible ones because these constraint conditions are only sufficient conditions

to ensure the feasible solutions, but not necessary. This paper focuses on the domain knowledge of feasibility and studies the necessary and sufficient conditions to ensure that the neighbourhood solutions are feasible. On this basis, combined with other effective domain knowledge, a new neighbourhood structure is proposed, and the corresponding fast calculation method is given. Finally, the advantages of the proposed neighbourhood structure are verified by experiments.

## 4 Necessary and Sufficient Conditions for Feasible Neighbourhood Solutions

In this section, by analyzing the causes of infeasible solutions in the JSP, we give the necessary and sufficient conditions to judge the feasibility of neighbourhood solutions and prove their sufficiency and necessity. At the same time, we compare the proposed conditions with the existing conditions for feasible neighbourhood solutions and design a new neighbourhood structure by combining the proposed conditions with the existing domain knowledge. Finally, a quick calculation method to determine the feasibility of neighbourhood solutions is given in this section.

### 4.1 Analysis of the Causes of Infeasible Solutions

Unlike other shop scheduling problems, there exist infeasible solutions to the job-shop scheduling problem, which makes the problem much more challenging than the others. In this part, we will analyze the causes of infeasible solutions through the disjunctive graph in the JSP. As mentioned above, if a closed-loop appears in the disjunctive graph of a solution in JSP, the solution must be infeasible. However, whether all infeasible JSP solutions have closed loops in the disjunctive graph is uncertain and needs further discussion. Therefore, Proposition 7 is introduced and proved below:

**Proposition 7.** *In the job-shop scheduling problem, the necessary and sufficient condition for an infeasible solution is the existence of a closed-loop in the disjunctive graph of the solution.*

**Proof.** The sufficiency of Proposition 7 has been explained before, so only the necessity of this proposition will be given here. It is assumed that there is an infeasible solution whose disjunctive graph does not have a closed loop. Since this solution is infeasible, there is at least one operation whose start and end time cannot be calculated. We can call the corresponding vertex  $x$ . In the disjunctive graph of the JSP, every vertex, except  $s$  (start node), has at least one predecessor (another vertex, which points to  $x$ ). If the start time and end times of  $x$  cannot be calculated, at least one predecessor cannot be calculated either.

Since there is no loop in the disjunctive graph, the vertex that cannot be calculated will eventually transfer to  $s$ . However, since  $s$  is a virtual node, its start and end times are 0. Since  $s$  is a computable vertex,  $x$  is also a computable vertex. It is inconsistent with the hypothesis.

Although the conclusion of Proposition 7 seems obvious, the necessary condition is often overlooked by scholars. In the following part, we will demonstrate the necessary and sufficient conditions for feasible neighbourhood solutions according to the conclusion of Proposition 7.

#### 4.2 Proof of Sufficiency and Necessity

According to Proposition 7, we can know that in the JSP, if the disjunctive graph of a solution exists in at least one closed loop, the solution is infeasible; if not, the solution is feasible. Therefore, the problem of sufficient and necessary conditions for feasible neighbourhood solutions can be transformed into sufficient and necessary conditions for forming a new acyclic disjunctive graph by changing the direction of the disjunctive arc from an acyclic disjunctive graph. It should be noted that the change in the direction of the disjunctive arc here is not arbitrary, but corresponds to the change in the processing sequence of the operations on the machine. However, the transformed problem is still very complex. Since the neighbourhood solution is feasible or infeasible, the necessary and sufficient conditions for feasible solutions are completely opposite to those for infeasible solutions. From this, we can further transform the problem into a necessary and sufficient condition for a looped disjunctive graph by changing the direction of the disjunctive arc from an acyclic disjunctive graph. When the final transformation problem is solved, the original problem can also be solved accordingly.

**Proposition 8.** *Two operations,  $u$  and  $v$ , are processed on the same machine, and  $u$  processes before  $v$ . When a neighbourhood solution is generated by changing the position of  $v$  to the position just before  $u$ , the necessary and sufficient condition to form a loop in the disjunctive graph of the neighbourhood solution is that there exists a path from  $u$  to  $jp[v]$  in the disjunctive graph of the current solution.*

**Proof.** (1) Sufficiency: if there exists a path from  $u$  to  $jp[v]$  in the disjunctive graph of the current solution, the path will still exist in the neighbourhood solution because there is nothing changed about  $jp[v]$ . Then the loop  $jp[v]-v-u-jp[v]$  exists in the disjunctive graph. (2) Necessity: if the neighbourhood solution is infeasible, there must exist a loop in the disjunctive graph of this

solution according to Proposition 7, and  $u$  and  $v$  must be in the loop because there is no change in any other operations. Note the loop as  $v-u-p-v$ , where  $p$  is a path in the disjunctive graph of the neighbourhood solution. It is easy to know that only the  $jp[v]$  and  $w$  (the same operation as  $mp[u]$  in the current solution) connect  $v$  in the neighbourhood solution. If  $jp[v]$  is not in the  $p$ , then  $w$  must be in the  $p$ . However, due to  $p$  also existing in the current solution, there must exist a loop  $u-p-u$ , which conflicts with the premise that the current solution is feasible. Therefore,  $jp[v]$  must be in  $p$ , and there must exist a path from  $u$  to  $jp[v]$  in the current solution.

**Proposition 9.** *Two operations,  $u$  and  $v$ , are processed in the same machine, and  $u$  processes before  $v$ . When a neighbourhood solution is generated by changing the position of  $u$  to the position just after  $v$ , the necessary and sufficient condition to form a loop in the disjunctive graph of the neighbourhood solution is that there exists a path from  $js[u]$  to  $v$  in the disjunctive graph of the current solution.*

**Proof.** (1) Sufficiency: if there exists a path from  $js[u]$  to  $v$  in the disjunctive graph of the current solution, the path will still exist in the neighbourhood solution because there is nothing changed about  $js[u]$ , and then the loop  $u-js[u]-v-u$  exists in the disjunctive graph. (2) Necessity: if the neighbourhood solution is an infeasible one, there must exist a loop in the disjunctive graph of this solution according to Proposition 7, and  $u$  and  $v$  must be in the loop because there is no change in any other operations. Note the loop as  $v-u-p-v$ , where  $p$  is a path in the disjunctive graph of the neighbourhood solution. It is easy to know that  $u$  only connects to the  $js[u]$  and  $w$  (the same operation as  $ms[v]$  in the current solution) in the neighbourhood solution. If  $js[u]$  is not in the  $p$ , then  $w$  must be in the  $p$ . However, due to  $p$  also existing in the current solution, there must exist a loop  $v-p-v$ , which conflicts with the premise that the current solution is feasible. Therefore,  $js[u]$  must be in  $p$  and there must exist a path from  $js[u]$  to  $v$  in the current solution.

According to Proposition 8, we can know that whether there is a path from  $u$  to  $jp[v]$  in the current solution is the key to whether a loop exists in the disjunctive graph of a neighbourhood solution, generated by moving  $v$  to the position just before  $u$  from the initial solution. Similarly, according to Proposition 9, we can know that whether there is a path from  $js[u]$  to  $v$  in the current solution is the key to whether a loop exists in the disjunctive graph of a neighbourhood solution, generated by moving  $u$  to the position just after  $v$  from the initial solution. With these, it is easy to know the necessary and sufficient conditions, which do not exist paths from  $u$  to  $jp[v]$  or from

$js[u]$  to  $v$  in the current solution, for feasible neighbourhood solutions in the JSP.

### 4.3 Difference between Feasibility Conditions

To better understand the necessary and sufficient conditions mentioned here, they are compared with the conditions used in Propositions 5 and 6. Here we only use the condition in Proposition 5, which can generate feasible neighbourhood solutions by moving  $v$  to the position just before  $u$ , as an example.

The constraint condition is as follows: when  $F(u) + p_u \geq F(jp[v]) + p_{jp[v]}$  is satisfied, move  $v$  to the position just before  $u$  will not generate infeasible solutions. In the current solution, there may be one of three relationships among operations between  $u$ ,  $v$  and  $jp[v]$ , as shown in Figure 9. The known determination condition is that operation  $u$  is processed before the operation  $v$ , and the operation  $jp[v]$  is processed before the operation  $v$ , so the operation  $u$  to the operation  $v$  and the operation  $jp[v]$  to the operation  $v$  are represented by solid lines in Figure 9. However, the relationship between  $u$  and  $jp[v]$  is unclear, so it has three possibilities and is represented by dotted lines.

- (1) For the first possibility as Figure 9(a), there exists a path from  $u$  to  $jp[v]$ . It means  $u$  must be processed before  $jp[v]$  and the numerical constraints between these two operations are  $F(u) + p_u < F(jp[v]) + p_{jp[v]}$ , which is not satisfied with the numerical constraints in Proposition 5.
- (2) For the second possibility as Figure 9(b), there exists

a path from  $jp[v]$  to  $u$ . It means  $jp[v]$  must be processed before  $u$ , and the numerical constraints between these two operations are  $F(u) + p_u \geq F(jp[v]) + p_{jp[v]}$ , which is satisfied with the numerical constraints in Proposition 5. (3) For the third possibility as Figure 9(c), since there is no path connection between  $u$  and  $jp[v]$ , the completion time of  $u$  may be larger or smaller than or equal to the completion time of  $jp[v]$ . According to the analysis of the three situations above, only the second possibility and part of the third possibility which is satisfied with the numerical constraints  $F(u) + p_u \geq F(jp[v]) + p_{jp[v]}$ , can be regarded as feasible neighbourhood solutions by Proposition 5. However, with the necessary and sufficient conditions, we can know that all the neighbourhood solutions generated in the third possibility are feasible, and here is the difference between the necessary and sufficient conditions and the conditions in Proposition 5. Something similar occurs when a neighbourhood solution is generated by moving  $u$  to the position just after  $v$ .

The scheduling scheme in Figure 3 is used to explain further the limitations of the condition in Proposition 5, and it is shown again in Figure 10. In the current solution,  $O_{3,3}$  and  $O_{1,2}$  are in the same critical block and  $F(O_{3,3}) + p_{O_{3,3}} < F(jp[O_{1,2}]) + p_{jp[O_{1,2}]}$ , which does not satisfy with the condition in Proposition 5. Then the neighbourhood solution, which moves  $O_{1,2}$  to the position just before  $O_{3,3}$ , is considered as an infeasible solution in Proposition 5. However, according to Proposition 8, this neighbourhood solution is feasible, and the

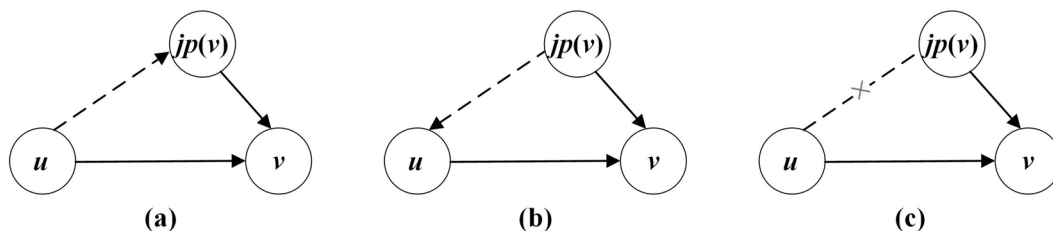


Figure 9 Three relationships among operations between  $u$ ,  $v$  and  $jp[v]$

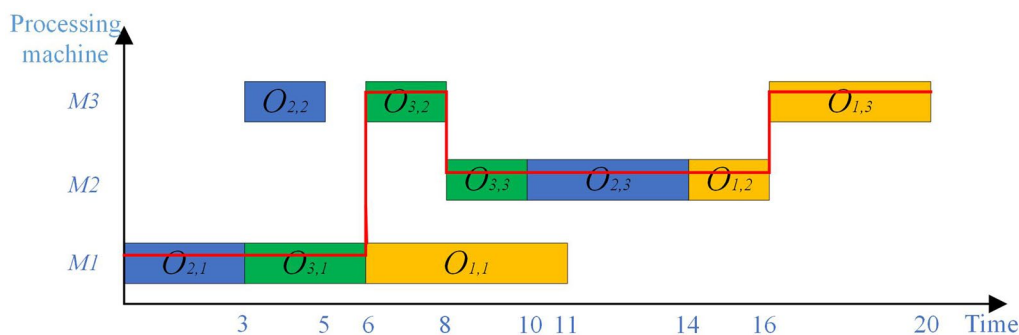


Figure 10 The Gantt chart of the current solution



Gantt chart can be shown in Figure 11. Also, it is easy to know that the makespan of the current solution and the neighbourhood solution are 20 and 19, which means that the condition in Proposition 5 may miss some better solutions.

Combining the necessary and sufficient conditions mentioned above with the domain knowledge in Proposition 1, 2 and 3, a new neighbourhood solution NNS can be described as follows:

*NNS:*  $u$  and  $v$  are two operations on the same critical block, and at least one of them is the first or last operation in the critical block. When there is no path from  $u$  to  $jp[v]$  in the disjunctive graph of the current solution, the neighbourhood solution can be generated by moving  $v$  to the position just before  $u$ ; or when there is no path from  $js[u]$  to  $v$  in the disjunctive graph of the current solution, a feasible neighbourhood solution can be generated by moving  $u$  to the position just after  $v$ .

#### 4.4 Calculation for Feasible Neighbourhood Solutions

By using the sufficient and necessary conditions proposed in this paper, it is necessary to determine whether there is a path between two vertices in the disjunctive graph of the current solution, which can be obtained through the Floyd-Warshall algorithm or Johnson algorithm. However, the efficiency of the Floyd-Warshall algorithm or Johnson algorithm is not satisfactory in this problem. Therefore, a path matrix is proposed to indicate whether there is a path between any two vertices in a disjunctive graph. Unlike the adjacency matrix, the path matrix can represent the relationship between non-adjacent nodes. The pseudocode for calculating the path matrix is as follows:

---

#### Algorithm 1 A path matrix Generator

---

```

1: let matrix  $P$  be a  $|V| \times |V|$  array and initialized to 0
2: for  $k$  from 1 to  $|V|$  do
3:   for  $t$  from 1 to  $|V|$  do
4:     if ( $P[t][mp[k]]=1 \parallel P[t][jp[k]]=1$ ) then
5:        $P[t][k]=1$ 
6:     end if
7:   end for
8:    $P[mp[k]][k]=1$ 
9:    $P[jp[k]][k]=1$ 
10: end for
    
```

---

$|V|$  is the number of vertices in the graph, and if  $P[t][k]=1$ , it means that there exists a path from  $t$  to  $k$ . Through the pseudocode, we can see that the time complexity of path matrix calculation is  $O(|V|^2)$ , which is smaller than the complexity of Floyd-Warshall algorithm  $O(|V|^3)$  and Johnson algorithm  $O(|V|^2 \log |V|)$ . However, if the size of the problem is large, such calculations are still expensive. To further save calculation time, we proposed a fast calculation method, which does not directly calculate whether there is a path between two vertices but uses a mixed method to find all feasible neighbourhood solutions of the current solution. The neighbourhood solution generated by moving  $u$  to the position just after  $v$  is used as an example.

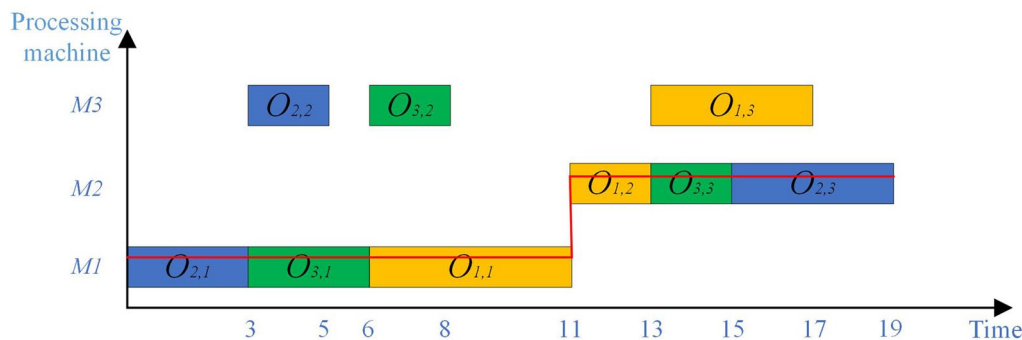


Figure 11 The Gantt chart of the neighbourhood solution

**Algorithm 2** A fast calculation method for feasible solutions

---

```

1: if  $R(v) + p_v \geq R(js[u]) + p_{js[u]}$  then
2:   return 0
3: end if
4: let  $L$  be a list, and the list is empty and let  $x=js[u]$ 
5: while (1) then
6:   if  $(js[x]=v \parallel ms[x]=v)$  then
7:     return 1
8:   end if
9:   if  $R(v) + p_v < R(js[x]) + p_{js[x]}$  then
10:    put  $js[x]$  into  $L$ 
11:   end if
12:   if  $R(v) + p_v < R(ms[x]) + p_{ms[x]}$  then
13:    put  $ms[x]$  into  $L$ 
14:   end if
15:   if  $L$  is empty then
16:     return 0
17:   else then
18:      $x$  = the first element in  $L$  and delete it from  $L$ 
19:   end if
20: end while

```

---

This algorithm is easy to understand based on the analysis of the constraints above. If the value returned is 1, it means that the generated neighbourhood solution is infeasible; if the value returned is 0, it means that the generated neighbourhood solution is feasible. Although, in essence, this method is still to judge whether there is a path between two nodes, the time needed to calculate will be significantly reduced after combining constraints. The algorithm used to judge the neighbourhood solution generated by moving  $v$  to the position just before  $u$  is similar to before.

## 5 Tabu Search Algorithm for JSP

In this section, the tabu search (TS) algorithm is introduced and used to test the neighbourhood structures. The algorithm uses a tabu list to avoid duplicate searches, and it can escape from local optima by proper move selection strategy. The  $i$ -TSAB for JSP proposed by Ref. [8] achieved great success. Besides this, the TS and TS/SA proposed by Zhang et al. [9, 10] also performed very well. This paper uses the strategy in TS

proposed by Zhang et al. [9] to test the neighbourhood structures.

### 5.1 Initial Solution

The initial solution has a certain influence on the optimization results of TS, but it does not mean that the better the fitness value of the initial solution is, the better the fitness value of the searched solution is, but that the position of the initial solution in the whole solution space has a certain influence on the optimization results. However, because the JSP is a complex discrete optimization problem, it is difficult to determine where the initial solution in the solution space has the best effect on the optimization. Therefore, the random generation method is used to obtain an initial solution in this paper.

### 5.2 Proposed Neighbourhood Structure

As mentioned above, this paper proposes a novel neighbourhood structure using domain knowledge for local search in the JSP. The domain knowledge used to ensure that the neighbourhood solution is feasible is the necessary and sufficient condition found in this paper. At the same time, a fast calculation method of feasible neighbourhood solution is given, which makes the algorithm efficient and sufficient for local search.

### 5.3 Evaluation of Neighbourhood Solutions

In order to reduce the computing time of neighbourhood solutions, the estimation strategy proposed by Balas and Vazacopoulos is used in the algorithm [16]. The core of this method is to calculate the parameters of all operations that change the processing order in the neighbourhood solution and select the maximum value as the estimated value. Suppose that the operations that change the processing order are  $\{O_1, O_2, \dots, O_t\}$ , and then the estimated value is  $\max\{F'(O_i) + p_{O_i} + R'(O_i)\}$ , where  $i=\{1, \dots, t\}$  [16, 29]. The empirical testing showed that this estimation method reduces 20% to 40% computing time than the exact approach. In addition, the experiment results showed that the optimal results of using approximate evaluation are not worse than the optimal results obtained using accurate decoding [9].

### 5.4 Tabu List and the Selection of Neighbourhood Solutions

The tabu list and the selection of neighbourhood solutions are vital points of the TS. The tabu list, including the tabu contents and length, is used to avoid duplicate

searches. The selection of neighbourhood solutions indicates the evolutionary direction of the algorithm, and it is determined by the quality of the neighbourhood solutions and the tabu content previously. In this paper, the tabu list and selection strategy refer to the content in Ref. [9]. More details can be found in the relevant literature.

### 5.5 Termination Criterion

The algorithm stops when satisfied with any condition below: (1) the number of iterations reaches a maximum value; (2) the number of disimproving iterations reaches a maximum value; (3) the solution is proved optimal. The parameters of the specific termination criteria will be specified in different experiments later.

## 6 Computational Experiments and Analysis

In this section, the experiments on the neighbourhood structure with necessary and sufficient conditions proposed in this paper are presented to demonstrate the superiority of this new neighbourhood structure. The experiment is divided into three parts: (1) in the first part, the experiment compared the characteristics of different neighbourhood structures, including the number of neighbourhood solutions and the calculation time by each iteration; (2) in the second part, the experiment compared results from the proposed neighbourhood structure and the existing neighbourhood structures on instances of different sizes; (3) in the third part, the experiment compared the TS combined with the new neighbourhood structure with other local search algorithms, and the optimal solutions obtained by different algorithms in some classical instances are compared. The results of the same type of instances with the same size were combined to compare their mean relative errors. The size of the instance can be represented by  $(n \times m)$ , where  $n$  is the number of jobs and  $m$  is the number of machines. The algorithm ran in the VC++ language on a personal computer with a CPU i7-9750H processor (2.6 GHz). The 152 well-known benchmark problems taken from the literature were used to test the property of the neighbourhood structures, and they include the following classes [30]: (a) 40 instances denoted as LA01-40; (b) 80 instances denoted as TA01-80; (c) 10 instances denoted as ORB01-10, 10 instances denoted as SWV01-10, 5 instances denoted as ABZ5-9 and 4 instances denoted as YN1-4; d) 3 instances denoted as FT6, FT10, FT20. These instances can be found on the website: <http://optimizer.com/TA.php>.

### 6.1 The Characteristics of Different Neighbourhood Structures

In this part, the characteristics of different neighbourhood structures are compared. To be fair, the number of

iterations, instead of the number of disimproving iterations, was regarded as the termination criterion, and the number was 1 million. Only three neighbourhood structures, N5, N6, and N7, which performed better in the above neighbourhood structures, were selected compared with NNS. 120 instances from class (a) and class (b) of 15 different sizes were used in this experiment. To make the comparison of experimental results more prominent, the results of all neighbourhood structures are compared against the results of N5 because it generates the least number of neighbourhood solutions and costs the least calculation time by each iteration. In this paper,  $s\_num$  and  $T$  represent the ratio of the number of neighbourhood solutions generated and the time spent by the neighbourhood structure in an iteration to the corresponding results in N5. In NNS, the T1 and T2 indicate the time used for algorithm 1 and algorithm 2, respectively. The results are shown in Table 1. Since the instances of LA and TA can contain all different sizes of data sets, the experimental results of these two kinds of instances are only listed in Table 1.

From the experimental results of this part, we can find that NNS can obtain the most neighbourhood solutions in the local search process, and when the ratio of  $n/m$  is larger, the gap between the neighbourhood solutions obtained by NNS and those obtained by other neighbourhood structures is larger. This phenomenon occurs because the larger the  $n/m$  value is, the more operations are in each critical block. When other constraints are used to judge whether the neighbourhood solution is feasible, a large number of feasible solutions will be regarded as infeasible. Of course, using NNS will take more computation time. By using the method proposed in this paper, the computation time of large-scale problems can be greatly reduced, which makes the computation time of NNS within an acceptable range.

### 6.2 Results Compared with Other Neighbourhood Structures

Similar to the experiment in the first part, the termination criterion was 1 million iterations, and only N5, N6 and N7 were used to compare with NNS. 152 instances mentioned before were used in this part of the experiment. Each instance was run 10 times, and the best and the average makespan were presented. The mean relative error (MRE) was used here to show the gap between the optimal solution and the solution result, and it can be calculated by the formulation  $MRE = 100 \times (C - LB_{best}) / LB_{best}$ , where  $C$  presents the makespan in the experiment and  $LB_{best}$  presents the makespan of the optimal solution. The  $MRE_b$  is used to represent the MRE of the best solution, and the  $MRE_{av}$  is used to represent the mean performance. The results are shown in Table 2.

**Table 1** The comparison of the number of neighbourhood solutions and the time spent

Instance	Size	N5		N6		N7		NNS		
		s_num	T	s_num	T	s_num	T	s_num	T1	T2
LA01-05	10×5	1.00	1.00	3.24	1.82	5.67	3.03	10.83	5.55	4.46
LA06-10	15×5	1.00	1.00	5.03	3.59	14.63	6.69	30.41	13.15	11.07
LA11-15	20×5	1.00	1.00	8.27	3.97	20.01	7.13	40.90	13.33	11.25
LA16-20	10×10	1.00	1.00	1.70	1.23	2.30	1.46	4.17	2.83	2.16
LA21-25	15×10	1.00	1.00	2.29	1.30	3.59	1.81	7.63	4.58	3.29
LA26-30	20×10	1.00	1.00	3.26	1.76	6.26	2.62	14.98	7.28	4.75
LA31-35	30×10	1.00	1.00	5.60	2.50	17.09	4.89	40.67	17.07	9.35
LA35-40	15×15	1.00	1.00	1.53	1.16	2.22	1.42	4.55	4.20	2.37
TA01-10	15×15	1.00	1.00	1.48	1.21	1.88	1.43	3.37	4.20	2.38
TA11-20	20×15	1.00	1.00	1.56	1.27	2.15	1.60	4.35	6.29	3.10
TA21-30	20×20	1.00	1.00	1.43	1.21	1.79	1.43	3.44	7.86	2.55
TA31-40	30×15	1.00	1.00	1.70	1.33	2.50	1.74	5.59	13.15	4.09
TA41-50	30×20	1.00	1.00	1.53	1.27	2.11	1.59	4.73	13.56	3.50
TA51-60	50×15	1.00	1.00	3.14	2.04	7.26	3.28	14.74	33.50	7.88
TA61-70	50×20	1.00	1.00	1.75	1.41	2.12	1.91	7.33	35.42	4.85
TA71-80	100×20	1.00	1.00	4.72	1.76	11.21	1.95	23.81	208.54	6.51

**Table 2** The comparison of the results from different neighbourhood structures

Instance	Size	N5		N6		N7		NNS	
		MRE <sub>b</sub>	MRE <sub>av</sub>	MRE <sub>b</sub>	MRE <sub>av</sub>	MRE <sub>b</sub>	MRE <sub>av</sub>	MRE <sub>b</sub>	MRE <sub>av</sub>
LA01-05	10×5	0.00	0.81	0.00	0.00	0.00	0.00	0.00	0.00
LA06-10	15×5	0.00	0.78	0.00	0.00	0.00	0.00	0.00	0.00
LA11-15	20×5	0.00	0.11	0.00	0.00	0.00	0.00	0.00	0.00
LA16-20	10×10	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
LA21-25	15×10	0.31	0.54	0.00	0.13	0.00	0.17	0.00	0.16
LA26-30	20×10	0.71	1.30	0.45	0.74	0.52	0.59	0.10	0.58
LA31-35	30×10	0.00	0.53	0.00	0.00	0.00	0.00	0.00	0.00
LA35-40	15×15	0.20	0.66	0.19	0.44	0.10	0.41	0.08	0.32
TA01-10	15×15	0.54	0.98	0.25	0.55	0.25	0.57	0.21	0.47
TA11-20	20×15	1.46	1.99	0.96	1.42	0.84	1.64	0.65	1.27
TA21-30	20×20	2.90	3.67	2.41	3.03	2.72	3.16	2.29	2.93
TA31-40	30×15	1.33	1.86	0.92	1.36	1.02	1.94	0.70	1.24
TA41-50	30×20	5.46	6.30	4.40	5.27	4.72	5.56	3.84	4.86
TA51-60	50×15	0.00	0.21	0.00	1.22	0.02	0.89	0.00	0.30
TA61-70	50×20	0.08	0.52	0.06	0.51	0.16	0.90	0.10	0.50
TA71-80	100×20	0.00	0.20	0.10	0.90	0.03	0.57	0.00	0.06
ORB01-10	10×10	0.04	0.38	0.07	0.22	0.06	0.24	0.02	0.22
SWV01-05	20×10	1.61	2.59	1.37	2.59	1.30	2.70	1.77	3.26
SWV06-10	20×15	6.71	8.06	5.72	7.58	6.12	7.56	6.77	8.34
ABZ05-06	10×10	0.00	0.11	0.00	0.02	0.00	0.02	0.00	0.02
ABZ07-09	20×15	2.55	3.08	1.83	2.27	1.78	2.14	1.49	2.07
YN01-04	20×20	5.03	5.49	4.64	5.12	4.63	5.34	4.40	5.21
FT06	6×6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FT10	10×10	0.00	0.52	0.00	0.34	0.00	0.39	0.00	0.31
FT20	20×15	0.00	0.24	0.00	0.60	0.00	0.97	0.00	0.56



In this part of the experiment, we compared four different neighbourhood structures. From the experimental results, it is easy to know that the neighbourhood structure proposed in this paper is obviously superior to other existing neighbourhood structures. To further illustrate, this paper selected four instances and drew box diagrams under different neighbourhood structures, as shown in Figure 12, which showed the superiority of the neighbourhood structure proposed in this paper.

### 6.3 Comparison with Other Local Search Algorithms

Although the focus of this paper is not on the design of the new algorithm for the JSP, in order to further illustrate the effectiveness of the new neighbourhood structure, this paper will use TS with NNS to compare with the results listed in Ref. [31], which are also non-population based meta-heuristic algorithms. The algorithms mentioned in the paper contain *i*-TSAB [8], TS/SA [10], TS/PR [12], and IEBO [31]. The instances used here were YN1-4, SVW01-10, and TA01-50, which are the most difficult ones. The termination criterion was the number

of disimproving iterations, which was 1 million. In addition, if the current optimal solution is not updated after 100,000 iterations, a solution will be randomly generated as the current solution of the next iteration. Each instance was run 10 times. The best and the average results were presented. The  $MRE_b$  is used to represent the MRE of the best solution, and the  $MRE_{av}$  is used to represent the mean performance. The results are shown in Table 3.

As seen from the experimental results in Table 3, although the results in this paper are slightly inferior to the best results in existing studies, the differences are very small. It should be noted that the improved algorithm based on tabu search is used in other literature, but only the tabu search algorithm is used in this paper, which illustrates the effectiveness of the neighbourhood structure in this paper from another aspect. The specific results of each instance are shown in Table 4, where  $C_b$  and  $C_a$  mean the best and average results, and  $T$  means the mean calculation time.

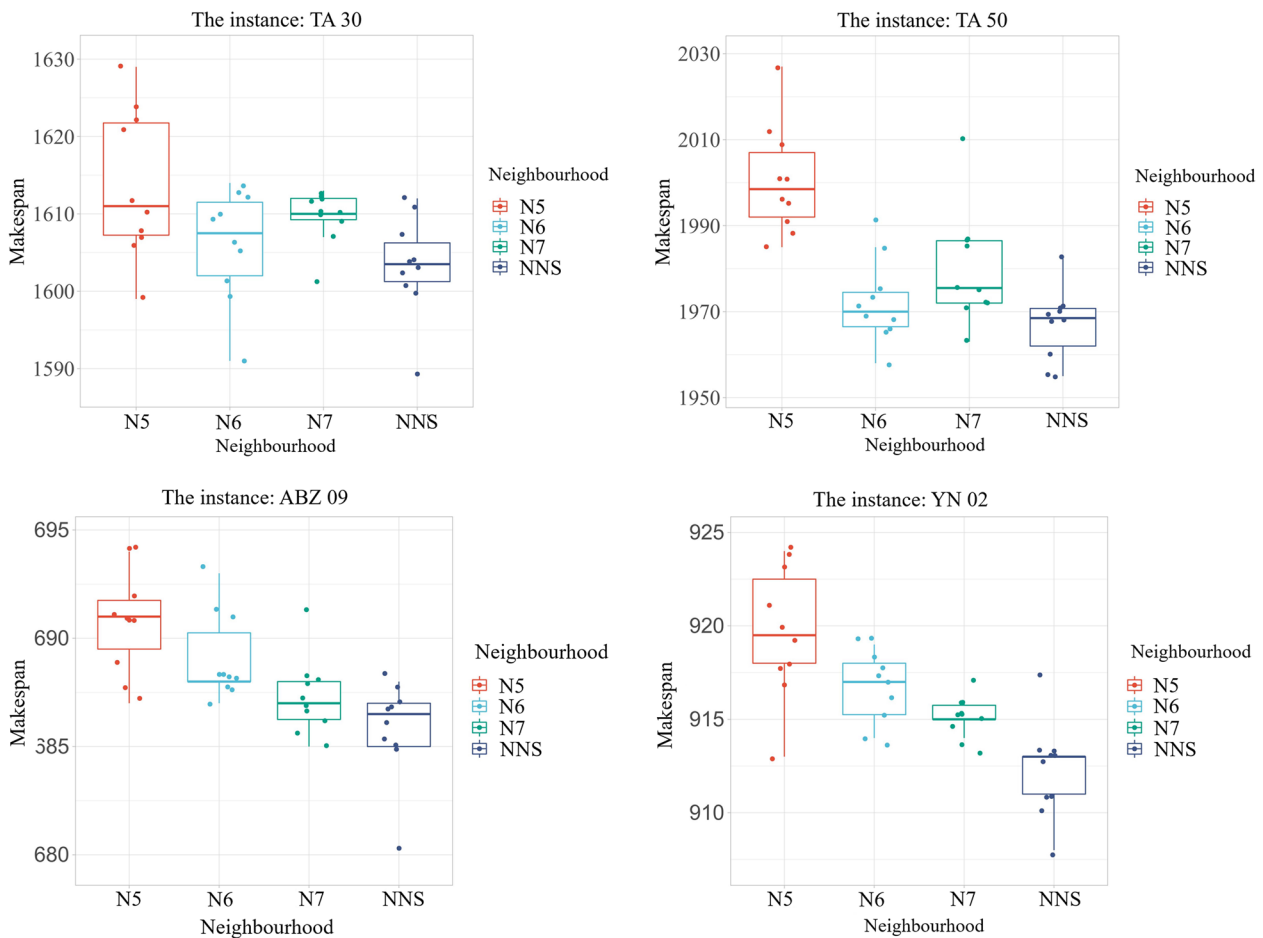


Figure 12 Box diagrams for four instances

**Table 3** The comparison with other local search algorithms

Instance	Size	i-TSAB	TS/SA		TS/PR		IEBO		TS(NNS)	
		$MRE_{av}$	$MRE_b$	$MRE_{av}$	$MRE_b$	$MRE_{av}$	$MRE_b$	$MRE_{av}$	$MRE_{av}$	$MRE_{av}$
TA01-TA10	15×15	0.00	0.01	0.11	0.01	0.01	0.00	0.01	0.01	0.07
TA11-TA20	20×15	0.22	0.25	0.79	0.14	0.38	0.14	0.22	0.16	0.49
TA21-TA30	20×20	1.98	1.97	2.49	1.98	2.19	1.88	2.01	1.99	2.39
TA31-TA40	30×15	0.34	0.37	0.75	0.22	0.40	0.25	0.38	0.58	0.73
TA41-TA50	30×20	3.22	3.17	3.93	2.84	3.34	2.90	3.30	3.68	3.93
SVW01-05	20×10	–	0.17	1.14	0.10	0.23	0.11	0.35	0.27	0.35
SVW06-10	20×15	–	4.25	5.95	3.79	4.33	3.74	4.34	4.29	4.39
YN1-4	20×20	–	3.98	4.51	3.87	4.10	3.87	4.02	4.23	4.50

**Table 4** The specific results of each instance

Instance	$LB_{best}$	$C_b$	$C_a$	$T(s)$	Instance	$LB_{best}$	$C_b$	$C_a$	$T(s)$
TA01	1231	1231	1231	102	TA 33	1788	1804	1808.5	888
TA 02	1244	1244	1244.1	175	TA 34	1828	1832	1832.7	1241
TA 03	1218	1218	1218.3	162	TA 35	2007	2007	2007	339
TA 04	1175	1175	1176.8	198	TA 36	1819	1819	1819.4	758
TA 05	1224	1224	1224.2	219	TA 37	1771	1780	1786.3	834
TA 06	1238	1238	1239.8	265	TA 38	1673	1673	1675.9	1161
TA 07	1227	1228	1228	108	TA 39	1795	1795	1796.8	967
TA 08	1217	1217	1217	125	TA 40	1651	1687	1690.4	1121
TA 09	1274	1274	1276.7	196	TA 41	1906	2028	2031.8	1219
TA 10	1241	1241	1241.9	163	TA 42	1884	1959	1963.7	1174
TA 11	1357	1357	1359.9	554	TA 43	1809	1868	1874.6	1201
TA 12	1367	1367	1370.6	364	TA 44	1948	1992	1999.1	1155
TA 13	1342	1342	1348.1	157	TA 45	1997	2001	2002.8	1350
TA 14	1345	1345	1345	148	TA 46	1957	2029	2032.4	1300
TA 15	1339	1339	1344.7	361	TA 47	1807	1913	1917.6	1325
TA 16	1360	1360	1361.8	347	TA 48	1912	1968	1971.7	1072
TA 17	1462	1462	1470.1	364	TA 49	1931	1981	1985.8	1156
TA 18	1377	1398	1403.9	361	TA 50	1833	1936	1943.2	774
TA 19	1332	1333	1340.7	596	SWV01	1407	1409	1410	236
TA 20	1348	1348	1350.7	297	SWV02	1475	1478	1480.7	223
TA 21	1642	1642	1647.7	359	SWV03	1398	1407	1407.6	207
TA 22	1561	1600	1610.7	185	SWV04	1464	1465	1466	255
TA 23	1518	1560	1564.3	409	SWV05	1424	1428	1428.6	253
TA 24	1644	1645	1652.3	376	SWV06	1630	1675	1676.4	271
TA 25	1558	1595	1597.4	438	SWV07	1513	1600	1600.7	232
TA 26	1591	1650	1654.5	533	SWV08	1671	1770	1772.3	319
TA 27	1652	1680	1687.1	277	SWV09	1633	1662	1662.6	243
TA 28	1603	1603	1614.7	311	SWV10	1663	1750	1753	357
TA 29	1583	1629	1631	110	YN01	854	887	889.2	180
TA 30	1528	1588	1595.9	141	YN02	870	907	910.5	162
TA 31	1764	1764	1764	658	YN03	859	894	897	189
TA 32	1774	1809	1814.9	889	YN04	929	972	973.5	260

## 6.4 Discussion

In this paper, the performance of NNS is verified by three parts of the experiments. In the first part of the experiment, the characteristics of the NNS neighbourhood structure are highlighted by comparing it with the existing neighbourhood structure. Compared with N5, N6, and N7 neighbourhood structures, NNS can obtain the most feasible neighbourhood solutions, especially for problems with large  $n/m$  values. Of course, NNS takes more computation time, but by using the hybrid method proposed in this paper, the computation time can be greatly reduced. In the second part of the experiment, 152 examples were solved by using the same algorithm configuration in different neighbourhood structures. By comparing the experimental results, we can see that the NNS neighbourhood structure has advantages. The third part of the experiment uses TS with NNS to compare with other local search algorithms. The experimental results show that although only the basic tabu search algorithm is used in this paper, the gap between the results from the tabu search algorithm with the proposed neighbourhood structure and the best results in the existing literature is very small, which further demonstrates the effectiveness of NNS.

## 7 Conclusion and Future Work

In this paper, the reason for generating infeasible solutions in the JSP is analyzed, and the sufficient and necessary conditions to ensure feasible neighbourhood solutions are proposed and proved. A new neighbourhood structure, NNS, is proposed by combining the proposed conditions and the existing domain knowledge. After that, a fast calculation method for this neighborhood structure. Compared with the existing famous neighbourhood structures, NNS can obtain more feasible neighborhood solutions with a small computational time increase. In addition, the new neighborhood structure shows better performance in most of the instances, which further shows the superiority of NNS. In future research, the following aspects are worthy of further study:

- 1) Extending NNS to other scheduling problems: Since the problems, like flexible job-shop scheduling problems and job-shop scheduling problems with different constraints, also have infeasible solutions, and the reasons for them are the same as in this paper. Therefore, the work in this paper can be extended to these scheduling problems.
- 2) Clipping the neighborhood solutions: Although the neighbourhood structure in this paper can obtain the most feasible neighbourhood solutions, a large part

of them are not improved on the current solutions. Therefore, these unimproved neighborhood solutions can be removed by constraint to improve the efficiency of local search.

- 3) Analysing the fitness landscape of the JSP: Most of the research on the JSP is focused on the mining and application of local domain knowledge, but the research on the global domain knowledge, such as the fitness landscape, is relatively few. Therefore, exploring the fitness landscape for the JSP is worthy of further study.

### Acknowledgements

Not applicable.

### Author contributions

LG made contributions to the conception, experiment, and writing; XL supervised the whole work and substantively revised it; LG made contributions to the design of the work; CW made contributions to the conception, and experiment. All authors read and approved the final manuscript.

### Authors' Information

Lin Gui, born in 1995, is currently a Ph.D. candidate at *State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, China*. He received his bachelor degree from *Shandong University, China*, 2018. His main research interests are shop scheduling and algorithm optimization.

Xinyu Li, born in 1985, is currently a professor at *State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, China*. He received his Ph.D. degree in industrial engineering from *Huazhong University of Science and Technology, China*, in 2009. His main research interests are intelligent manufacturing systems, shop scheduling, intelligent optimization and machine learning.

Liang Gao, born in 1974, is currently a professor at *State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, China*. He received his Ph.D. degree in mechanical engineering from *Huazhong University of Science and Technology, China*, in 2002. His main research interests are intelligent optimization method and its application in design and manufacturing.

Cuiyu Wang, born in 1983, is currently a Ph.D. candidate at *State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, China*. She received her master degree from *Huazhong University of Science and Technology, China*. Her main research interests are shop scheduling and algorithm optimization.

### Funding

Supported by National Natural Science Foundation of China (Grant Nos. U21B2029 and 51825502).

### Data availability

The data that support the findings of this study are openly available in <http://optimizer.com/TA.php>.

### Declarations

#### Competing Interests

The authors declare no competing financial interests.

Received: 24 December 2022 Revised: 15 June 2023 Accepted: 25 June 2023

Published online: 10 August 2023

## References

- [1] M M Ahmadian, A Salehipour, T C E Cheng. A meta-heuristic to solve the just-in-time job-shop scheduling problem. *European Journal of Operational Research*, 2021, 288(1): 14-29.
- [2] P Zou, M Rajora, S Y Liang. A new algorithm based on evolutionary computation for hierarchically coupled constraint optimization: methodology and application to assembly job-shop scheduling. *Journal of Scheduling*, 2018, 21(5): 545-563.
- [3] G Al Aqel, X Y Li, L Gao. A modified iterated greedy algorithm for flexible job shop scheduling problem. *Chinese Journal of Mechanical Engineering*, 2019, 32: 21.
- [4] L Gui, L Gao, X Y Li. Anomalies in special permutation flow shop scheduling problems. *Chinese Journal of Mechanical Engineering*, 2020, 33: 46.
- [5] H Xiong, S Shi, D Ren, et al. A survey of job shop scheduling problem: The types and models. *Computers & Operations Research*, 2022, 142: 105731.
- [6] O H Constantino, C Segura. A parallel memetic algorithm with explicit management of diversity for the job shop scheduling problem. *Applied Intelligence*, 2022, 52(1): 141-153.
- [7] Y An, X Chen, K Gao, et al. Multiobjective flexible job-shop rescheduling with new job insertion and machine preventive maintenance. *IEEE Transactions on Cybernetics*, 2022, 53(5): 3101-3113.
- [8] E Nowicki, C Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 2005, 8(2): 145-159.
- [9] C Y Zhang, P G Li, Z L Guan, et al. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 2007, 34(11): 3229-3242.
- [10] C Y Zhang, P G Li, Z L Guan, et al. A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & Operations Research*, 2008, 35(1): 282-294.
- [11] X Y Li, J Xie, Q J Ma, et al. Improved gray wolf optimizer for distributed flexible job shop scheduling problem. *Science China Technological Sciences*, 2022, 65(9): 2105-2115.
- [12] B Peng, Z P Lü, T C E Cheng. A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, 2015, 53: 154-164.
- [13] C R Vela, S Afsar, J J Palacios, et al. Evolutionary tabu search for flexible due-date satisfaction in fuzzy job shop scheduling. *Computers & Operations Research*, 2020, 119: 104931.
- [14] S Mahmud, A Abbasi, R K Chakraborty, et al. Multi-operator communication based differential evolution with sequential Tabu Search approach for job shop scheduling problems. *Applied Soft Computing*, 2021, 108: 107470.
- [15] J Błażewicz, W Domschke, E Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 1996, 93(1): 1-33.
- [16] E Balas, A Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 1998, 44(2): 262-275.
- [17] J Xie, X Y Li, L Gao, et al. A new neighbourhood structure for job shop scheduling problems. *International Journal of Production Research*, 2022, 61(7): 2147-2161.
- [18] Q Luo, Q Deng, G Gong, et al. An efficient memetic algorithm for distributed flexible job shop scheduling problem with transfers. *Expert Systems with Applications*, 2020, 160: 113721.
- [19] M S Islam, M P Nepal, M Skitmore, et al. A knowledge-based expert system to assess power plant project cost overrun risks. *Expert Systems with Applications*, 2019, 136: 12-32.
- [20] C N Potts. Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research*, 1980, 28(6): 1436-1441.
- [21] M Abedi, R Chiong, N Noman, et al. A multi-population, multi-objective memetic algorithm for energy-efficient job-shop scheduling with deteriorating machines. *Expert Systems with Applications*, 2020, 157: 113348.
- [22] G Zhang, L Zhang, X Song, et al. A variable neighborhood search based genetic algorithm for flexible job shop scheduling problem. *Cluster Computing*, 2019, 22: 11561-11572.
- [23] Z Shao, W Shao, D Pi. Effective heuristics and metaheuristics for the distributed fuzzy blocking flow-shop scheduling problem. *Swarm and Evolutionary Computation*, 2020, 59: 100747.
- [24] P J Van Laarhoven, E H Aarts, J K Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 1992, 40(1): 113-125.
- [25] M M Nasiri, F Kianfar. A GES/TS algorithm for the job shop scheduling. *Computers & Industrial Engineering*, 2012, 62(4): 946-952.
- [26] J Xie, X Y Li, L Gao, et al. A hybrid algorithm with a new neighborhood structure for job shop scheduling problems. *Computers & Industrial Engineering*, 2022, 169: 108205.
- [27] W Li, D Han, L Gao, et al. Integrated production and transportation scheduling method in hybrid flow shop. *Chinese Journal of Mechanical Engineering*, 2022, 35: 12.
- [28] S K Zhao. Research on multi-operation joint movement neighbourhood structure of job shop scheduling problem. *Journal of Mechanical Engineering*, 2020, 56(13): 192-206. (in Chinese)
- [29] L Gui, X Y Li, L Gao, et al. An approximate evaluation method for neighbourhood solutions in job shop scheduling problem. *IET Collaborative Intelligent Manufacturing*, 2022, 4(3): 157-165.
- [30] J J Van Hoorn. The current state of bounds on benchmark instances of the job-shop scheduling problem. *Journal of Scheduling*, 2018, 21(1): 127-128.
- [31] Y Nagata, I Ono. A guided local search with iterative ejections of bottleneck operations for the job shop scheduling problem. *Computers & Operations Research*, 2018, 90: 60-71.

Submit your manuscript to a SpringerOpen<sup>®</sup> journal and benefit from:

- Convenient online submission
- Rigorous peer review
- Open access: articles freely available online
- High visibility within the field
- Retaining the copyright to your article

---

Submit your next manuscript at ► [springeropen.com](https://www.springeropen.com)

---