A Neural Network-based Hybrid Method to Generate Feasible Neighbors for Flexible Job Shop Scheduling Problem

Aydin TEYMOURIFAR^{1,2,*}, Gurkan OZTURK^{1,2}

¹Faculty of Engineering, Anadolu University, Turkey

²Computational Intelligence and Optimization Laboratory (CIOL), Anadolu University, Turkey

Copyright ©2018 by authors, all rights reserved. Authors agree that this article remains permanently open access under the terms of the Creative Commons Attribution License 4.0 International License

Abstract In this paper, a hybrid method is proposed to generate feasible neighbors for the flexible job shop scheduling problem. Many of the optimization and artificial intelligence methods have been used to solve this important and NP-hard combinatorial problem which provide the basis for solving real-life problems. It is well-known that for such problems the hybrid methods obtain better results than the other approaches. For instance, the applied non-hybrid neural networks for the combinatorial problems, as the Hopfield neural network, usually converge early. Also, their results almost always contain large gaps. These shortcomings prevent them to find good results. Another necessity for a quality search is to find suitable neighbors of the obtained solutions, however, it is possible to create infeasible neighbors during the optimization process. The aim of this study is to overcome these deficiencies. In the suggested approach, at first, an initial solution is generated and then using the left shift heuristics, its gaps are removed. Based on the critical path and critical block concepts, 6 neighbors are constructed for the obtained solution. After the generation of each neighbor, a neural network runs and controls the constraints of the problem. If the achieved neighbor is feasible it is saved. Else if it is infeasible, the neural network tries to transform it into a feasible solution. This is done by applying penalties to the start time of the operations on the violated constraints, which shifts them to the right or the left. During this process, if there are not any violated constraints, the neural network reaches the stable condition so it stops and the obtained solution is saved as a feasible neighbor. Otherwise, after a certain number of the iterations, it stops without any feasible neighbors. Then these steps are repeated for the other created neighbors. This constraint-based process provides an effective and diverse search. Finally, the obtained neighbors, are improved using the left shift heuristics. Also to demonstrate the importance of the initial solutions, they are generated randomly and also using the Giffler and Thompson's heuristic. The comparison between the proposed approach and the methods from the literature shows that it constructs better neighbors. However, using the Giffler and Thompson heuristic to create the initial solution improves the results significantly.

Keywords Flexible Job Shop Scheduling, Hybrid Neural Network, Heuristics, Neighborhood Structures, Giffler and Thompson's Heuristic

1 Introduction

The flexible job shop scheduling problem (FJSSP) is one of the well-known NP-hard scheduling problems because of its application and theoretic aspects. Many of the real-life scheduling problems are its extensions. Previously, several methods based on the mathematical optimization and the computational intelligence-based approaches have been proposed in the literature to solve the FJSSP. Actually, there aren't any strict differences between these approaches and most of the applied neural networks (NNs) for the FJSSP are formulated as the mixed integer programming models. The Hopfield neural network (HNN) that is applied for solving many of the combinatorial problems is one of them. In some

studies of the literature, NN constructs a solution, whereas, in some others, it prevents infeasible solutions by checking the constraints. When a NN constructs solutions, generally some deficiencies like premature convergence and a large number of gaps in the achieved schedules occur. So it is necessary to use heuristics like the local and global left shifts to solve these troubles. Early convergence, prevents to achieve the global optimal results. When a NN is used as a preventive tool by checking prohibited moves of the operations by checking the constraints of the problem, the limited move set reduces the diversity of the search. All these reasons point out that in a non-hybrid approach, using NN cannot achieve good results. While combining them with the heuristics methods improves their performance with escaping from the local optimal points for the problems with high complexity.

Also, to provide a quality search, it is necessary to define effective neighborhood structures. Otherwise, a blind search will be done. Sometimes, during the process of the neighbor generation, infeasible results are obtained. In this study, a hybrid method that is easy to implement is proposed that constructs feasible neighborhoods for the FJSSP. Different from the previous works, the proposed NN is used to transform infeasible neighbors to the feasible ones considering the constraints of the problem. The general steps of this approach can be summarized as that at first, an initial solution is generated, then the local left shift heuristic is used to convert the solution to a semi-active solution. Two approaches are used to construct the initial solutions. In the first approach, they are created randomly, while the Giffle and Thompson's (GT) algorithm is used to generate them in the second approach. The critical paths and critical blocks of the initial solution are found and the neighbors are created by applying 6 different types of moves to the operations on them. These moves are applied even if some of them are forbidden according to the constraints of the problem. Due to the fact that the operations on these structures determine the makespan, the changes made on them significantly affect the objective function. Also, using more than one neighborhood structure leads to a better quality result. When a neighbor is constructed, the NN runs and controls the constraints of the problem, which are related to the start time of the operations and the order of them on the jobs and the machines. If the solution is feasible, it is saved, otherwise, penalties are applied to the violated constraints for adjusting them. During this process, the operations of the violated constraints are shifted to the left or the right. This means that their start time decreases or increases. These changes can lead to new violations, so after each iteration, the neural network attempts to correct the constraints that are violated again. Meanwhile, if a feasible solution is obtained, it is saved. Applying all moves, including the infeasible ones and then adjusting the violations, increases the diversity of search. After a pre-defined number of the iterations, if any feasible solution is not obtained, the NN stops. If a feasible neighbor is achieved, it transforms into a more effective solution by applying the global and local left shifts. Thereafter, all of these steps are repeated for the neighbors constructed by the neighborhood generating unit.

In the second section of the article, the FJSSP definition and its benchmarks are described. Literature review, the NNs presented for the scheduling problems in previous studies, description of the proposed algorithm and its design steps, implementation and the obtained results, comparison and discussion are the next sections of the article. Conclusion and future works are explained in the last section.

2 Problem Definition and Test Instances

In the FJSSP there is a set of n jobs ($J = \{J_1, J_2, \dots, J_n\}$), each consists of n_i operations ($O_{j1}, O_{j2}, \dots, O_{jn_i}$) that plan to be processed on a subset of m machines ($M = \{M_1, M_2, \dots, M_m\}$) as $M_{j1}, M_{j2}, \dots, M_{jn_i}$ [43]. Other assumptions about the problem are: (i) the number of jobs and operations are known in advance, (ii) assignable machines and processing time of the operations on them are predefined, (iii) the start times and the processing times of all operations are integer, (iv) jobs are independent of each other, (v) interruption during the process of operations isn't allowed, (vi) each job has a consecutive set of operations according to the sequence constraints, (vii) each machine can process at most one operation at any time according to the resource constraint, (viii) setup time of the machines are negligible, (ix) each machine can work independently from the other machines with the maximum performance and there aren't any maintenance and breakdown times [39][46].

In this paper, O_{jik} stands for operation i of job j to be processed on machine k with R_{jik} release time, ST_{jik} start time, P_{jik} processing time and C_{jik} completion time. The objective function is defined as to minimize the makespan, which is formulated in Equation 1.

$$\text{Minimize } \max_j(C_{n_j,j}) \quad (1)$$

The constraints of the problem are as Equations 2, 3, 4 and 5. According to the start time constraint, each operation

can start at a positive time that is greater than its release time.

$$ST_{ijk} \geq r_{ij} \geq 0 \quad (2)$$

As stated in the assumption of the problem, according to the sequence constraint, the operations of each job must be processed in a certain order. So as seen in Equation 3, if operation O_{ij} and $O_{i(j-1)}$ are assigned to machines l and k respectively, then the difference between their start times must be at least as much as $P_{i(j-1)k}$ that is the processing time of $O_{i(j-1)}$ on machine k .

$$ST_{ijl} - ST_{i(j-1)k} - P_{i(j-1)k} \geq 0 \quad (3)$$

Due to that each machine can process at most one operation at any time and interruption of the processes is forbidden, if an operation is not in the first order on the assigned machine, its start time must be after the finish time of the previous operation on the machine. So if operation O_{ij} starts before operation O_{pq} on machine k then the related resource constraint is Equation 4.

$$ST_{pjk} - ST_{ijk} - P_{ijk} \geq 0 \quad (4)$$

But if operation O_{ij} starts on machine k after the completion time of operation O_{pq} then the resource constraint is like Equation 5.

$$ST_{ijk} - ST_{pjk} - P_{pjk} \geq 0 \quad (5)$$

Only one of Equations 4 and 5 are valid at a time based on assignments of the operations [14][34][42].

There are several test instances for the FJSSP like Behnke and Geiger [5], Brandimarte [7], Hurink et al. [19], Dautère-Pérès and Paulli [10], Chambers and Barnes [9], Kacem et al. [20], Fattahi et al. [13] instances. If all machines of a problem can process all of the operations, that is a FJSSP with total flexibility. In the partial FJSSP, each operation can be processed on a subset of the machines set [20][44]. Partial FJSSPs are more difficult than the Total FJSSP [4].

3 Neural Networks Application for Solving Scheduling Problems in the Literature

One the most applied neural networks is the Hopfield neural network (HNN). After the first use of this method for an optimization problem by Hopfield, its hybrid types have been successfully applied for many of the problems in the communication, the pattern recognition and the combinatorial optimization. The analog to the digital conversation problem, the traveling salesman problem (TSP), the optimal process plan selection, the terminal assignment problem (TAP), the frequency assignment problem (FAP), the assignment of cells to switches in a cellular mobile network, the scheduling problems and the linear and non-linear programming problems are some examples of these problems [18][21][23][24][26][30][31][32][36][38][47]. The HNN is a dynamic and recurrent network in which each state is dependent on the previous state. The Boltzman machine, which can be considered as the stochastic type of the HNN, is very suitable for solving the scheduling problems. For instance, Fnaiech et al. have proposed a HNN for the production scheduling problem with taking account the different random maintenance periods [14]. Wang and Yang have summarized these NNs deficiencies as they are generally non-adaptive networks, with the predefined biases and the weights of the connection. Generally, the NNs cannot convergence to the valid solutions and don't locate in the global minimum and have poor scaling properties due to the use of the quadratic energy functions [41][47]. There are some studies about defining more proper energy functions. Foo et al. have presented a modified Tank-Hopfield linear programming network for solving mixed integer linear programming. They have defined a linear energy function that is more practical than the quadratic energy function. However, their algorithm needs more processing elements [15]. Simon and Takefuji have proposed an integer linear programming neural network (JLPNN) based on the Tank and HNN modifications. They have reported that there are several integer linear programming models for the resource scheduling problems but they are impractical in general. So they have presented a special structure of the integer linear programming as NN that seeks to minimize start

times of all jobs according to the constraints [34][35].

Real-life scheduling problems of the production systems about the staff, resource and the maintenance process are generally stochastic and dynamic problems with many constraints [1][17][29]. So artificial learning methods application is a useful approach to solve such problems. Even notwithstanding the shortcomings of the NNs in the combinatorial problems, using them combined with the other optimization methods causes to good results. Ming and Mak have presented a hybrid HNN combined with a genetic algorithm (GA) that achieves quality solutions. The energy function (or the Lyapunov function) of this HNN is specially designed for solving their problem which is the process plan selection problem. The HNN generates initials solutions and the GA search for the better solutions, hereby the algorithm achieves near global optimal results [24]. Salcedo-Sanz and Yao have presented a hybrid HNN to solve the terminal assignment (TA) and cells to switches in a cellular mobile network problem. The HNN controls the constraints of the problem and the GA searches for better solutions [31][32]. In a similar study, Sancho Salcedo-Sanz et al. have suggested a hybrid HNN combined with a simulated annealing algorithm (HopSA), for the frequency assignment problem (FAP) for the satellite communication systems [30]. Tang and Zhang have proposed a NN combined with heuristics for the flow shop scheduling problem with sequence dependent setup times [37]. Yu and Liang have suggested a NN and GA based hybrid algorithm for the expanded job shop scheduling problem (EJSSP), in which the GA optimizes the sequences and the NN adjusts the start times [49].

The left shifts are commonly used to remove the gaps that occur during schedule generation. The left shift of activity i in schedule S with start time ST_i is a shift that transforms schedule S into a feasible schedule as S' with start time ST'_i where $ST'_i < ST_i$ and $ST'_j = ST_j$, for all other activities. If $ST_i - ST'_i = 1$, the shift is an one period left-shift. A local left-shift is a shift that can be achieved by successive one-period left shifts. A global left shift is a left shift that isn't a local left shift. With the local left shift, some operations of the schedule can start earlier without changing the order of the other operations and with the global left shift, some operations can start earlier without other operations delaying. In a feasible schedule, if there aren't any possible local left shifts, it is a semi-active schedule and if for all activities neither local shift nor global left shift can be performed, it is an active schedule. A non-delay schedule is a feasible schedule that for all activities neither local shift nor global left shift can be performed even if the activities can be preempted at integer time points [2][8][12][22] [33][50]. The optimal schedule is an active schedule but not necessary a non-delay schedule [45].

Figure 1. Classification of the schedules [27]

Many algorithms use the GT algorithm for solving the scheduling problems, in one or more stages as the initial solution creation or as a part of a meta-heuristic or heuristic methods. It is a systematic recursive procedure to generate all or some of the active schedules. With some modifications, this algorithm can construct the non-delay schedules. It isn't a complex procedure, while it is a useful method, because it restricts the search space effectively to the active or the non-delay schedules spaces [27].

NNs are also used for improving the neighborhood search methods of the scheduling problems [51]. In the graph theory, the neighborhoods and the local search are related to the critical operations, the critical path and the critical blocks. The longest path from the first operation to the last one is called the critical path and the operations on the critical path are the critical operation. Makespan is equal to the length of the critical path. It is possible to decompose the critical path

into blocks, that is the maximal sequence of adjacent critical operations on the same machine [16][25][40]. Several neighborhood structures are defined for the scheduling problems [48]. For example, in N_1 structure, one machine is chosen and all possible swaps for the jobs that are assigned to this machine are applied. In N_2 , two machines are chosen and all possible swaps of the jobs from these different machines are considered. In N_3 , randomly a machine and a job that is not assigned to it are chosen. Then the selected job is transferred to a valid position on the chosen machine [3][11][28]. In the N_4 neighborhood structure, all operations of the critical block are moved to the very beginning or very end of the block. Nowicki and Smutnicki have proposed the N_5 neighborhood structure that is smaller than the N_1 and N_4 , which searches for new neighbors on the borders of the blocks. The N_6 neighborhood structure switches a schedule to another schedule with interchanging two operations, which are on the same block of the critical path [6][11][25].

4 The Proposed Method

The general steps of the proposed method are summarized in Algorithm 1, where at first, an initial solution is generated, then it is transformed into a semi-active schedule by applying local left shifts. Then 6 neighbors of this solution are generated according to Algorithm 2. The changes are made based on the start time of the operations instead of the order of them, then the NN runs to check and adjust infeasible neighbor. The structure of the applied NN is similar to the NNs in the literature for the scheduling problems [45]. It has the units which stand for the start time, the sequence and the resource constraints of the operations. Each unit has an input (I), an activation function (AC) and an output (O). The start time neurons are as much as the number of operations, which is equal to $n \times m$. The bias value of these neurons is zero. The start times of the operations in the random solution are the input values of these neurons. The activation function is like Figure 2 and Equation 6 that is to prevent the start times less than the release times of the operations. The output of this activation function is a feasible value for the operations start time.

Figure 2. The activation function of the start time neurons

$$O_{ST_{ij}} = \begin{cases} r_{ij}, & \text{if } I_{ST_{ij}} < r_{ij} \\ I_{ST_{ij}}, & \text{if } r_{ij} < I_{ST_{ij}} \end{cases} \quad (6)$$

There is a priority relationship between the operations of each job. For example, as seen in Figure 3, if O_8 and O_7 are from the same job, Equation 7 must be valid, which means that the start time of O_8 must be greater than the start time of O_7 at least as much as the processing time of O_7 on the assigned machine. Because according to the sequence constraint, the start time of O_8 must be after the finish time of O_7 . The connections that are black in Figure 3, and their weights (+1 or -1) represent the coefficient of the start times in the sequence constraint that is as Equation 7.

$$ST_8 - ST_7 - P_{7k} \geq 0 \quad (7)$$

The activation function of the sequence constraint unit is as Figure 6 and Equation 10. If there is a violated sequence constraint, the value of the corresponding neuron is negative and the constraint is added to the list of the violated constraints. The total number of the sequence constraints equals to $nm - n$.

Figure 3. A sequence constraint neuron that controls the priority relation between O_7 and O_8

According to the resource constraint that specifies the order of operations on the machines, a machine cannot process more than one job at any time. Therefore, only after that an operation is finished, the other operation can start. The number of these constraints is $nm - m$. Suppose that O_2 and O_6 are sideways on machine m . If O_6 precedences O_2 , then the relation between the neuron of the corresponding constraint is as Figure 4 and Equation 8.

$$ST_2 - ST_6 - P_{6m} \geq 0 \quad (8)$$

Figure 4. The resource constraint neuron when O_6 is finished before O_2

But if O_2 precedences O_6 , then the structure of the neurons are like Figure 5 and Equation 9.

$$ST_6 - ST_2 - P_{2m} \geq 0 \quad (9)$$

Figure 5. The resource constraint neuron when O_2 is finished before O_6

By applying the activation function that is as in Figure 5 and Equation 10, to the input value of the sequence and resource neurons, the output value of the violated neurons becomes negative. Hereby the set of the violated constraints is formed.

Figure 6. The activation function of the sequence and resource constraints units

$$O_{C_i}(t) = \begin{cases} 0, & \text{if } I_{C_i}(t) \geq 0 \\ -I_{C_i}(t), & \text{if } I_{C_i}(t) < 0 \end{cases} \quad (10)$$

The neurons corresponding to the violating sequence constraints must be adjusted using the feedback weights seen in Figure 7 as the black arrows. This means that the start time of the corresponding operations decreases or increases, and the operations are shifted to the left or the right on the time axis.

Figure 7. The adjustment connections and their weights

If P and V stand for the value of the penalty and violation, the adjustment value is $P \times V$, as it is in Equation 11, which is done with taking the start time constraints into account, so these constraints will not be violated during this process. In this way, the start times of the violated constraints at the next iteration ($t+1$) are formed based on their values in the current iteration (t).

$$ST_i(t+1) = ST_i(t) \pm P \times V \quad (11)$$

After a similar adjustment process for the neurons related to the resource constraints, the NN passes to the next iteration. At the end of each iteration, the resource constraints are updated because it is possible to change the order of the operations on the machines. Also, new violations may occur during the adjustment process. So it is necessary to control all of the constraints unless the start time constraints, which are not violated. The iteration number of the NN is defined as 2500, but when a feasible solution is found, the NN stops and similar steps are repeated for the next neighbor (N_{i+1}). Also, if there isn't any feasible result for N_i after the last iteration, the algorithm passes to N_{i+1} . However, the NN generally converges before than 100 iterations. When these processes are finished for the last neighbor (N_6), the algorithm totally stops and the best neighbor is selected between the obtained ones. The general steps of the NN are described in Algorithms 3.

4.1 Illustrative example

Suppose that a random solution is created as it is in Figure 9 for the problem that is summarized in Figure 8. The first row shows the order of the operations while the second row shows the assigned machines.

		M ₁	M ₂	M ₃
Job 1	O ₁	2		3
	O ₂		2	
	O ₃		1	1
Job 2	O ₄	2	1	
	O ₅		3	
	O ₆	1	2	
Job 3	O ₇	3		2
	O ₈	2		

Figure 8. A FJSSP

Operations order	1	4	7	8	2	5	3	6
Assigned machine	3	2	3	1	2	2	3	2

Figure 9. A random solution

The corresponding schedule for this solution is shown in Figure 10, which is not a semi-active schedule and contains some gaps.

Figure 10. The corresponding schedule for the random solution

With applying some local left shifts, the random solution transforms into the schedule seen in Figure 11. The black line shows the critical path.

Figure 11. The achieved semi-active schedule with applying local left shifts to the random schedule

The corresponding NN to the random solution is as in Figure 12.

Figure 12. Corresponding NN to the random solution

As a simple move, as in Figure 13, the start time of O_5 changes to 1, then due to that the order on machine 2 is changed the structure of the resource constraint neurons (RC) are changed as in Figure 14. In the new state, all of the constraints are checked and it is noticed that the resource constraint between operations O_5 and O_2 is violated. So in the corresponding NN, that is depicted in Figure 14, the output of $RC_{5,2}$ is negative.

Figure 13. Infeasible neighbor of the random solution

Figure 14. Corresponding NN to the infeasible solution

As seen in Figure 15, in the next iteration, the NN shifts O_2 to the right but in this case the sequence constraint between this operation and O_3 is violated.

Figure 15. The NN applies a right shift to O_2

O_3 is shifted to the right and as seen in Figure 16, a feasible neighbor is obtained, so the NN stops and the algorithm passes to the next constructed neighbor and repeats the previously stated steps.

Figure 16. The NN applies a right shift to O_3

Finally, since the obtained schedule is not semi-active, a left shift is applied that improves the results as shown in Figure 17.

Figure 17. A feasible and semi-active neighbor of the random solution

5 Experimental Study and the Performance of the Suggested Approach

The described method is implemented in MATLAB 2013a on a system with a Pentium dual-core 2.20 GHz processor and 3.00 GB RAM. Two approaches are used to generate the initial solutions. In the first approach, they are created randomly, while the GT algorithm is used to construct them in the second approach. To show the effectiveness of the proposed method, it is compared with the local left shift and the global left shift heuristics based on the MK benchmarks from the literature. In the first approach, 10 random solutions are generated for each of the benchmarks and the results in Table 1 are the average of the results obtained by the algorithms.

Table 1. Comparison between the results obtained by the proposed approach and the left shifts, where the initial solutions are created randomly

Instances	Initial schedule	Local left shift	Global left shift	The proposed method
mk1	205.5	89.5	85.75	79.25
mk2	225.25	84.5	69	67
mk3	1493.5	428.5	308.75	299.75
mk4	418.5	142.5	122	119
mk5	718.75	292.75	252.5	246.25
mk6	723.25	226	168.75	160.5
mk7	1016.25	334.5	280.75	273.75
mk8	2814.5	835.5	662	659.5
mk9	2770	701	515	502
mk10	2542	625.5	431.5	427.5

In the second approach, 10 initial solutions are generated for each of the benchmarks according to the GT algorithm. The same procedures are applied to these solutions and the results in Table 2 are the average achieved results.

Table 2. Comparison between the results obtained by the proposed approach and the left shifts, where the initial solutions are created using the GT algorithm

Instances	Initial schedule	Local left shift	Global left shift	The proposed method
mk1	65.5	65.5	62.8	58.8
mk2	53.66	53.66	53.66	50.83
mk3	281	281	281	267
mk4	97	97	97	92.33
mk5	223	223	222	219.33
mk6	116.33	116.33	116.33	116.33
mk7	245.66	245.66	245.66	243
mk8	587.5	587.5	587.5	587.5
mk9	523.5	523.5	492	477
mk10	368	368	368	368

Brandimarte [7] using the constraint programming techniques has obtained the results which are shown in Table 3.

Table 3. Makespans obtained by the constraint programming-based algorithms [5]

Instances	The constraint programming
mk1	40
mk2	27
mk3	204
mk4	60
mk5	174
mk6	59
mk7	143
mk8	523
mk9	307
mk10	214

As seen in Tables 1 and 2, the proposed approach can generate good neighbors especially if the initial solutions are generated using the GT algorithm. Actually, these results are obtained by applying some changes on the initial solutions and not by a global optimization process. However, a comparison between these results and those are in Table 3 demonstrates that the achieved improvements are significant. Using this method as a part of the global optimization methods can improve the quality of search. Also, if it is applied for solving real-time scheduling problems, it can generate fair results in a little time, where the exact optimization methods application is hard and time-consuming.

6 Conclusion and Future Works

In this study, a hybrid method is proposed, which generates feasible neighbors for the FJSSP. It applies effective moves to the operations on the critical paths and the critical blocks, then by taking the constraints into account, constructs feasible solutions. It is well-known that effective results cannot be achieved when the neighbors are constructed based on the operations that are not on the critical paths and the critical blocks. Because in a schedule, the operations on the critical path specify the makespan. Also in the multi-objective models, with moving the operations on the critical path, the total workload and maximum workload of the machines also change. Before creating a neighbor for a random solution,

it should be converted to a semi-active or active schedule with the left shifts. Because it is not possible to find the critical paths and the critical blocks in a schedule that is not semi-active. It is possible to obtain infeasible solutions during the neighbor generation process. Actually, if the construction of the infeasible solutions is prevented, the quality and the diversity of the search may reduce. The NN in this work converts infeasible results into the feasible ones by applying penalties to the violated constraints. During this process, these operations are shifted to the right or to the left. In general, NN will converge sooner if only one of the operations are shifted to the right or to the left, as much as the violation value. The adjustment process may also result in the new violations, which must be adjusted in the next iterations. When the NN becomes stable, it means there are not any violated constraints and the achieved solution is feasible. But if after a certain number of iterations, the NN is not stable, a new neighbor is generated and the search continues. The proposed NN generally converges before 100 iterations. In the scheduling problems, usually, the solutions that are obtained by a NN contain gaps. If there are any gaps in the result, it is necessary to apply the left shifts to reduce the value of makespan.

Another important matter is that as in the global search, without decreasing the diversity, the space of the search should be appropriately restricted in the local search. So it can be a good approach to generate initial solutions and make changes to them using the heuristics such the GT algorithm that their results are near the optimal solutions.

Similar to the neighborhood structures, the proposed method transforms a solution into other solutions which are its neighbors. So it cannot be considered as a global search method. That is why it is compared with the left shift heuristics which are based on a similar approach. Actually, this method must be used combined with the global search methods, which improves their performance.

In future works, we plan to design a NN based on the mixed integer programming to solve the dynamic staff scheduling problem.

(The MATLAB codes would be accessed through the corresponding author's email.)

References

- [1] B. Afshar-Nadjafi, A. Rahimi, and H. Karimi. A genetic algorithm for mode identity and the resource constrained project scheduling problem. *Scientia Iranica*, 20(3):824–831, 2013.
- [2] C. Artigues, S. Demassey, and E. Neron. *Resource-constrained project scheduling: models, algorithms, extensions and applications*. John Wiley & Sons, 2013.
- [3] J. Behnamian, M. Zandieh, and S. F. Ghomi. Parallel-machine scheduling problems with sequence-dependent setup times using an aco, sa and vns hybrid algorithm. *Expert Systems with Applications*, 36(6):9637–9644, 2009.
- [4] D. Behnke and M. J. Geiger. Test instances for the flexible job shop scheduling problem with work centers. <https://www.deutsche-digitale-bibliothek.de/item/WG35B3KZZC4YAARHH3GSHGXEKWRZHY5L>.
- [5] D. Behnke and M. J. Geiger. Test instances for the flexible job shop scheduling problem with work centers. 2012.
- [6] J. Błażewicz, W. Domschke, and E. Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European journal of operational research*, 93(1):1–33, 1996.
- [7] P. Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research*, 41(3):157–183, 1993.
- [8] P. Brucker and S. Knust. *Complex scheduling (GOR-Publications)*. Springer, 2006.
- [9] J. B. Chambers and J. W. Barnes. Tabu search for the flexible-routing job shop problem. *Graduate program in Operations Research and Industrial Engineering, The University of Texas at Austin, Technical Report Series, ORP96-10*, 1996.
- [10] S. Dauzère-Pérès and J. Paulli. Solving the general multiprocessor job-shop scheduling problem. *Working Papers, Department of Mathematical Sciences, University of Aarhus*, 1995.

- [11] M. Dell'Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations research*, 41(3):231–252, 1993.
- [12] E. L. Demeulemeester and W. S. Herroelen. *Project scheduling: a research handbook*, volume 49. Springer Science & Business Media, 2006.
- [13] P. Fattahi, M. S. Mehrabad, and F. Jolai. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, 18(3):331–342, 2007.
- [14] N. Fnaiech, H. Hammami, A. Yahyaoui, C. Varnier, F. Fnaiech, and N. Zerhouni. New hopfield neural network for joint job shop scheduling of production and maintenance. In *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society*, pages 5535–5541. IEEE, 2012.
- [15] S. Y. Foo, Y. Takefuji, and H. Szu. Scaling properties of neural networks for job-shop scheduling. *Neurocomputing*, 8(1):79–91, 1995.
- [16] J. F. Gonçalves, J. J. de Magalhães Mendes, and M. G. Resende. A hybrid genetic algorithm for the job shop scheduling problem. *European journal of operational research*, 167(1):77–95, 2005.
- [17] M. Haghi, S. Fatemi Ghomi, and P. Hooshangi-Tabrizi. A novel deterministic model for simultaneous weekly assignment and scheduling decision-making in operating theaters. *Scientia Iranica*, 24(4):2035–2049, 2017.
- [18] J. J. Hopfield and D. W. Tank. “neural” computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.
- [19] J. Hurink, B. Jurisch, and M. Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum*, 15(4):205–215, 1994.
- [20] I. Kacem, S. Hammadi, and P. Borne. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 32(1):1–13, 2002.
- [21] M. P. Kennedy and L. O. Chua. Neural networks for nonlinear programming. *IEEE Transactions on Circuits and Systems*, 35(5):554–562, 1988.
- [22] R. Kolisch. *Project scheduling under resource constraints: efficient heuristics for several problem classes*. Springer Science & Business Media, 2013.
- [23] C.-K. Looi. Neural network methods in combinatorial optimization. *Computers & Operations Research*, 19(3):191–208, 1992.
- [24] X. Ming and K. Mak. A hybrid hopfield network-genetic algorithm approach to optimal process plan selection. *International Journal of Production Research*, 38(8):1823–1839, 2000.
- [25] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management science*, 42(6):797–813, 1996.
- [26] J. H. Park and H. Jeong. Solving the tsp using an effective hopfield network. In *Proc. IEEE Int. Conf. Neural Networks*, 1990.
- [27] M. L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media, 2002.
- [28] V. Roshanaei, B. Naderi, F. Jolai, and M. Khalili. A variable neighborhood search for job shop scheduling with set-up times to minimize makespan. *Future Generation Computer Systems*, 25(6):654–661, 2009.
- [29] S. Sadjadi, R. Soltani, M. Izadkhah, F. Saberian, and M. Darayi. A new nonlinear stochastic staff scheduling model. *Scientia Iranica*, 18(3):699–710, 2011.
- [30] S. Salcedo-Sanz, R. Santiago-Mozos, and C. Bousoño-Calzón. A hybrid hopfield network-simulated annealing approach for frequency assignment in satellite communications systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(2):1108–1116, 2004.
- [31] S. Salcedo-Sanz and X. Yao. A hybrid hopfield network-genetic algorithm approach for the terminal assignment problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(6):2343–2353, 2004.

- [32] S. Salcedo-Sanz and X. Yao. Assignment of cells to switches in a cellular mobile network using a hybrid hopfield network-genetic algorithm approach. *Applied Soft Computing*, 8(1):216–224, 2008.
- [33] C. Schwindt and J. Zimmermann. *Handbook on Project Management and Scheduling Vol. 1*. Springer, 2015.
- [34] F. Y.-P. Simon and T. Takefuji. Integer linear programming neural networks for job-shop scheduling. In *Neural Networks, 1988., IEEE International Conference on*, pages 341–348. IEEE, 1988.
- [35] F. Y.-P. Simon and Y. Takefuji. Stochastic neural networks for solving job-shop scheduling. i. problem representation. In *Neural Networks, 1988., IEEE International Conference on*, pages 275–282. IEEE, 1988.
- [36] K. Smith, M. Palaniswami, and M. Krishnamoorthy. A hybrid neural approach to combinatorial optimization. *Computers & operations research*, 23(6):597–610, 1996.
- [37] L. Tang and Y. Zhang. Heuristic combined artificial neural networks to schedule hybrid flow shop with sequence dependent setup times. In *International Symposium on Neural Networks*, pages 788–793. Springer, 2005.
- [38] D. Tank and J. Hopfield. Simple ‘neural’ optimization networks: An a/d converter, signal decision circuit, and a linear programming circuit. *IEEE Transactions on Circuits and Systems*, 33(5):533–541, 1986.
- [39] A. Temourifar and G. Ozturk. New dispatching rules and due date assignment models for job shop scheduling problems. *Accepted to International Journal of Manufacturing Research*, 2017.
- [40] C. R. Vela, R. Varela, and M. A. González. Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *Journal of Heuristics*, 16(2):139–165, 2010.
- [41] B. Widrow. Darpa neural network study. *AFCEA Int. Press, Fairfax, VA (February 1988)*, 1988.
- [42] T. Willems and J. Rooda. Neural networks for job-shop scheduling. *Control Engineering Practice*, 2(1):31–39, 1994.
- [43] W. Xia and Z. Wu. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering*, 48(2):409–425, 2005.
- [44] L.-N. Xing, Y.-W. Chen, P. Wang, Q.-S. Zhao, and J. Xiong. A knowledge-based ant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing*, 10(3):888–896, 2010.
- [45] S. Yang. An improved adaptive neural network for job-shop scheduling. In *SMC*, pages 1200–1205, 2005.
- [46] S. Yang and D. Wang. Constraint satisfaction adaptive neural network and heuristics combined approaches for generalized job-shop scheduling. *IEEE Transactions on Neural Networks*, 11(2):474–486, 2000.
- [47] S. Yang and D. Wang. A new adaptive neural network and heuristics hybrid approach for job-shop scheduling. *Computers & Operations Research*, 28(10):955–971, 2001.
- [48] M. Yazdani, M. Zandieh, R. Tavakkoli-Moghaddam, and F. Jolai. Two meta-heuristic algorithms for the dual-resource constrained flexible job-shop scheduling problem. *Scientia Iranica. Transaction E, Industrial Engineering*, 22(3):1242, 2015.
- [49] H. Yu and W. Liang. Neural network and genetic algorithm-based hybrid approach to expanded job-shop scheduling. *Computers & Industrial Engineering*, 39(3):337–356, 2001.
- [50] X. Yu and M. Gen. *Introduction to evolutionary algorithms*. Springer Science & Business Media, 2010.
- [51] M. Zandieh and M. Adibi. Dynamic job shop scheduling using variable neighbourhood search. *International Journal of Production Research*, 48(8):2449–2458, 2010.

Appendices

A Algorithms

Algorithm 1: Generate feasible neighbors

```

1 Input: Problem set;
2 Output: Feasible neighbors;
3 Create random solution;
4 Apply local left shift to the random solution;
5 Find the critical paths and critical blocks;
6 for  $i = 0; i < 6; i = i + 1$  do
7    $N_i \leftarrow$  Generate neighbor based on structure  $i$ ;
8   Control the constraints;
9   if  $N - i$  is feasible then
10    | Save it;
11  else
12    Run Neural Network for 100 times;
13    if there is a feasible solution then
14      | Apply local left shift to the feasible solution;
15      | Apply global left shift to the feasible solution;
16      |  $N_i \leftarrow$  The feasible solution;
17      | Exit from the Neural Network and Generate the next neighbor ( $N_{i+1}$ );
18    else
19      |  $N_i \leftarrow []$ ;
20    end
21  end
22 end
23 Calculate the objective functions;
24 Find the best neighbor between  $N_i$ ;

```

Algorithm 2: Neighborhood structures

```

1 Input: Problem set;
2 Output: Neighbors;
3 if  $i==1$  then
4   | Swap the operations on the same critical blocks;
5 else if  $i==2$  then
6   | Swap the operations of the critical path which are on different machines;
7 end
8 else if  $i==3$  then
9   | Insert the operations on the critical path to the different positions;
10 end
11 else if  $i==4$  then
12   | Swap the operations on the critical blocks with the first operation of the same critical block;
13 end
14 else if  $i==5$  then
15   | Swap the operations on the critical blocks with the last operation of the same critical block;
16 end
17 else if  $i==6$  then
18   | Swap the first and the last operations of the blocks;
19 end

```

Algorithm 3: Neural network

```

1 Input: neighbor;
2 Create the start time constraints;
3 Control the start time constraints;
4 Adjust the start time constraints;
5 Create the sequence constraints;
6 for  $i = 0; i < 100; i = i + 1$  do
7   Control the sequence constraints;
8   Adjust the sequence constraints;
9   Create the resource constraints;
10  Control the resource constraints;
11  Adjust the resource constraints;
12  Create the start time constraints;
13  Create the sequence constraints;
14  Create the resource constraints;
15  Control the start time constraints;
16  Control the sequence constraints;
17  Control the resource constraints;
18 end
19 Output: Feasible neighbor or [];

```