Firefly-Inspired Algorithm for Job Shop Scheduling

Joss Miller-Todd(✉), Kathleen Steinhöfel(✉), and Patrick Veenstra(✉)

Department of Informatics, King's College London,
30 Aldwych, Bush House, London WC2B 4BG, UK
{joss.miller-todd,kathleen.steinhofel,patrick.veenstra}@kcl.ac.uk

Abstract. Current research strongly suggests that hybrid local search algorithms are more effective than heuristics based on homogenous methods. Accordingly, this paper presents a new hybrid method of Simulated Annealing and Firefly Algorithm [SAFA] for the Job Shop Scheduling Problem (JSSP) with the objective of minimising the makespan. We provide an experimental analysis of its performance based on a set of established benchmarks.

Simulated Annealing [SA] continues to be a viable approach in identifying optimal and near optimal makespans for the JSSP. Similarly, recent deployments of the Firefly Algorithm [FA] have delineated its effectiveness for solving combinatorial optimisation problems efficiently. Therefore, the hybrid algorithm in question aims to combine the acclamatory strengths of SA and FA while overcoming their respective deficiencies.

Keywords: Job shop scheduling · Firefly · Simulated Annealing

1 Introduction

Scheduling deals with the allocation of resources to operations over given time periods, with the aim of optimising one or more objectives [16]. The Job Shop Scheduling Problem (JSSP) is one of many formulations in scheduling, and has attracted many researchers investigating properties of NP-hard problems and methods to tackle them. It is an important optimisation problem and, for the authors in particular, the problem that sparked, based on [20], the collaboration with Juraj Hromkovič and his research group, see [9,10,14].

There are many variants (relaxations and restrictions) of JSSP. In this paper, we consider the standard version defined as follows. There are n jobs in the set $J = \{j_1, j_2, \ldots, j_n\}$ and m machines $M = \{m_1, m_2, \ldots, m_m\}$. Each job $j \in J$ consists of m operations; one operation for each machine. Let $\pi_j(i)$ be the operation for job j performed on machine i. The operations π_j must be performed in a specific order. Let $\mathbf{O} \in [1, m]^{m \times n}$ contain the order of the operations for each of the jobs. The column vector $\mathbf{o}_j$ contains the order of operations for the job j. To be explicit, $o_{ij} \in [1, m]$ is the position of $\pi_j(i)$ in the sequence of operations for j. So if $o_{ij} = 4$, then operation $\pi_j(i)$ must be the 4-th operation

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 423–433, 2018.
https://doi.org/10.1007/978-3-319-98355-4_24

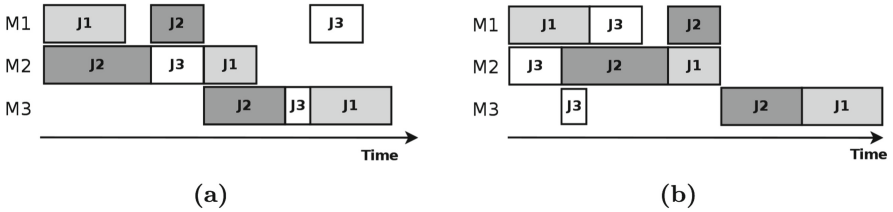

Fig. 1. Two possible schedules for an example instance of the job shop scheduling problem with three jobs on three machines, with times and orders defined in (1). The problem is to find the schedule that leads to the shortest makespan, that is, the shortest possible time needed to complete all operations.

to run for job j. Finally, the elements p_{ij} of the matrix $\mathbf{P} \in \mathbb{R}^{m \times n}$ denote the time taken for operation $\pi_j(i)$, $j \in J$, $i \in M$. Each machine can only process a single operation at a time, and once an operation has started, no preemption is permitted.

A schedule is a matrix $\mathbf{C} \in \mathbb{R}^{m \times n}$ containing the completion times of the operations. Element c_{ij} is the completion time of operation $\pi_j(i)$. Let the makespan L of $\mathbf{C}$ be $\max_{i,j} \mathbf{C}_{ij}$. The job shop scheduling problem is to find a valid schedule such that the makespan L (the total time taken for all jobs to complete) is minimised.

As an example, take a problem instance with three machines and three jobs, with the time matrix $\mathbf{P}$ and order matrix $\mathbf{O}$ as follows:

$$\mathbf{P} = \begin{array}{c} \begin{matrix} J_1 & J_2 & J_3 \end{matrix} \\ \begin{pmatrix} 3 & 2 & 2 \\ 2 & 4 & 2 \\ 3 & 3 & 1 \end{pmatrix} \end{array} \begin{matrix} M_1 \\ M_2 \\ M_3 \end{matrix} \qquad \mathbf{O} = \begin{array}{c} \begin{matrix} J_1 & J_2 & J_3 \end{matrix} \\ \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 1 \\ 3 & 3 & 2 \end{pmatrix} \end{array} \begin{matrix} M_1 \\ M_2 \\ M_3 \end{matrix} \tag{1}$$

Figure 1 shows two examples of valid schedules for this example instance of JSSP.

In this paper, we propose an algorithm for creating schedules that minimise the makespan of JSSP instances by combining the Simulated Annealing algorithm with a Firefly-inspired methodology. The rationale behind this combination is to escape from local minima and avoid becoming constrained in the solution space.

The JSSP has been a focal point for many researchers over the last few decades, primarily due to the growing need for efficiency accompanied by the rapid increase in the speed of computing [2]. This, together with the more recent developments in the availability of cloud resources allowing for large scale distributed and parallel computation, has opened up many additional avenues in terms of algorithmic techniques for working with optimisation problems.

The JSSP has a core motivation in production planning and manufacturing systems. Reduced costs and efficient production lines are a direct result of fast

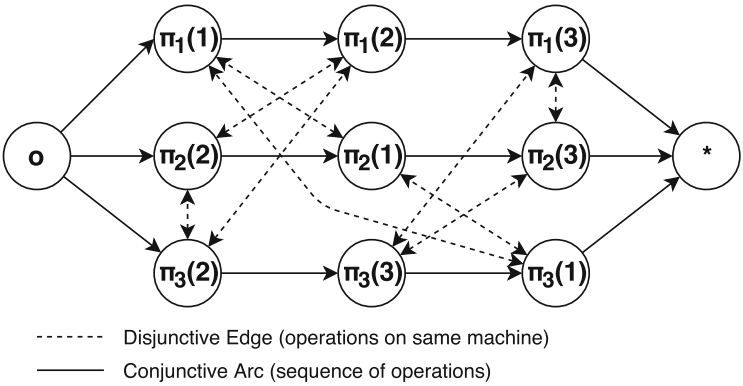

Fig. 2. Disjunctive graph exemplified from the previous section (Fig. 1). Recall that $\pi_j(i)$ refers to the operation for the j-th job on the i-th machine. Solid arcs indicate the order of operations for each job. Dashed edges indicate operations on the same machine.

and accurate scheduling solvers. Advancements in job shop scheduling also allow for prioritisation of operations and improved predicted completion times [1].

This paper builds on the research of Steinhöfel et al. on SA [20] and her work with Albrecht et al. on FA [13] by evaluating their symbiosis for the job shop scheduling problem. Moreover, the presented method is motivated by the property of a backbone structure, see [22], and the Firefly idea which enables the search of areas close to the estimated backbones of optimal schedules.

2 Background

The publication of a seminal paper by Johnson (1954) is largely credited as being the starting line for scheduling research [11]. The article outlines two and three-stage production schedules with the objective of minimising the total elapsed time. Early methods of solving JSSP instances include priority rules, branch and bound, integer programming, Monte-Carlo methods, stochastic analysis, algorithms with learning, and enumerative algorithms [15,17].

An important graphical representation of the JSSP, is the disjunctive graph proposed by Roy and Sussman [17].

Figure 2 shows a disjunctive graph for the example defined in the previous section. This graphical formulation provides an outline of all feasible schedules, enabling better interpretation than the usual Gantt chart representations. Modification of the schedule structure is achieved by activating disjunctive edges to represent an ordering of tasks on a machine path. Figure 3 outlines a concrete schedule equivalent to instance (a) in Fig. 1.

The source **o** and dummy vertex ***** act as start and end points respectively, allowing for full navigation of the directed graph; the longest path from **o** to ***** is the makespan. The makespan can be calculated by weighting all vertices by the

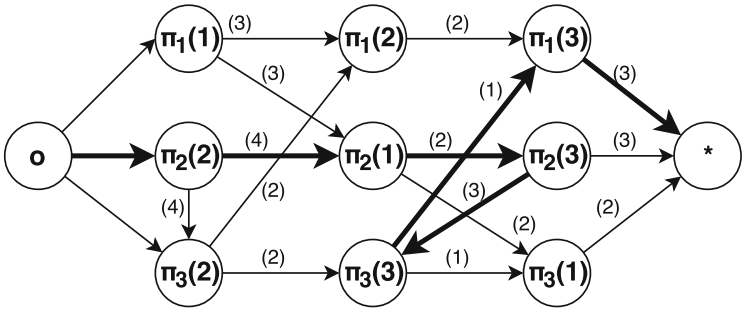

Fig. 3. Disjunctive graph of the schedule instance (a) from the previous section (Fig. 1). Recall that $\pi_j(i)$ refers to the operation for the j-th job on the i-th machine. The vertex weights are shown on their outgoing edges, and the thick arcs show a longest path with respect to vertex weights.

respective **P** values of each operation $\pi_j(i)$ and summing the total processing time of all operations on the longest path.

Note that, in [4], the graph derived from a disjunctive graph by considering a concrete schedule has exactly one arc for every disjunctive edge, let us call these *disjunctive arcs*. But any disjunctive arc shortcutting a directed path of disjunctive arcs does not contribute to a longest path (in terms of vertex weights) since the shortcut misses at least one vertex. Thus, we can omit these shortcuts from the graph.

Potts and Strusevich (2009) demonstrate a simple lower bound of JSSP instances by removing all disjunctive edges in such a graph and finding the maximum weight path in the resulting sub-graph [18]. Various more complex lower bounds are evaluated by dividing a JSSP instance into sub-problems by machine, calculating the lower bound for the single instance and selecting the optimal result.

2.1 Local Search

Local Search (LS) methods involve starting with some initial solution, and then iteratively moving to solutions that are similar, but have better objective function values. This iterative process repeats until a local optimum is found. A survey of LS methods for JSSP can be found in Vaessens et al. [24].

Let F be the set of feasible solutions for some problem instance. The cost of any $x \in F$ is defined by $c\colon F \to \mathbb{R}$. The neighbourhood function $N\colon F \to \{0,1\}^F$ determines which solutions in F are similar to x. That is, for $x \in F$, the solutions in the neighbourhood $N(x) \subseteq F$ are said to be neighbours to x. The execution of a local-search algorithm defines a walk in F such that each solution visited is a neighbour of the previously visited one. A solution $x \in F$ is called a local minimum with respect to a neighbourhood function N if $c(x) \leq c(y)$ for all $y \in N(x)$.

A major drawback of LS methods is that of becoming trapped in a local optimum. One attempt at addressing this problem is Simulated Annealing (SA). Annealing in metallurgy is the method of heating and then cooling metals to maximise the size of crystals. SA simulates this process for optimisation problems. As in LS methods, there is a neighbourhood function and a cost function. In SA, rather than always moving to a solution with a better objective value, there is some probability of moving to worse solutions. This probability is determined by the temperature, which reduces with time. A higher temperature means the algorithm is more likely to move to a worse solution. This allows the algorithm to leave local minima and explore more of the solution space.

In 1992, Laarhoven et al. first formulated the SA approach for the JSSP [25], along with further results by Blazewicz et al. [4], Steinhöfel et al. [20] and Satake et al. [19], respectively. SA has proven to be an effective technique for efficiently finding approximate results, overcoming to some extent the aforementioned limitations of LS.

2.2 Evolutionary Algorithms and Hybrids

Various evolutionary algorithms have been used as heuristics for solving JSSP instances, including Genetic Algorithms (GA) [7], Ant Colony Optimisation (ACO) [5] and Particle Swarm Optimisation (PSO) [6]. Instead of improving a single candidate solution, these meta-heuristics improve a population of solutions.

The method developed in this paper follows a line of research on hybrid approaches that combine SA with nature-inspired algorithms. Our method combines SA with the Firefly Algorithm (FA), which is one of the more recent nature inspired optimisation algorithms, published in 2008 by Xin-She Yang [6]. Fireflies are characterised by their luminescence, using their emittance of light to attract other fireflies - essentially their main form of communication.

The intensity of light emitted is directly proportional to the volume and rate at which other flies converge toward it. The attractiveness of the flies is linked to their objective function and monotonic decay of the attractiveness with distance, allowing individual flies to be attracted to any other firefly in the population, assuming they are close enough. The algorithm developed in this paper simplifies this approach somewhat, and resembles more accurately the algorithm developed by Steinhöfel et al. on protein structure prediction in lattice models [13]. This implementation considers a single target, setting the optimal instance as the beacon of light for the rest of the population. In this scenario, the so called fireflies all attempt to move in the same direction.

The rationale supporting the use of FA for the JSSP emanates from an understanding of the solution space, wherein schedule instances close to the optimal are more likely to have similar edge orientations. In addition, updates of the beacon during execution allow for the population to constantly change direction, increasing the likelihood of exploring optimal zones.

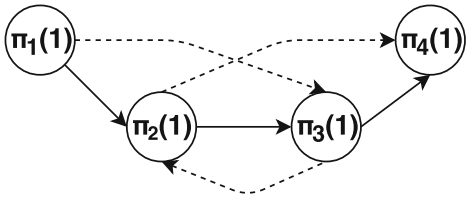

Fig. 4. Transition depicting edge flip on a machine path.

3 Our Algorithm

The method presented in this paper is a hybrid of Simulated Annealing (SA) and the Firefly Algorithm (FA).

The neighbourhood function determines for each solution x a subset of F. A solution belongs to that subset only if it can be reached from x with a single modification step, i.e., $N\colon F \to \{0,1\}^F$. That is, $N(x) \subseteq F$ is the set of feasible schedules similar to $x \in F$.

We define neighbours based on edge switching. Edges on a machine path can be switched such that a machine path $\pi_1(1)$-$\pi_2(1)$-$\pi_3(1)$-$\pi_4(1)$ can transition to $\pi_1(1)$-$\pi_3(1)$-$\pi_2(1)$-$\pi_4(1)$. A visual representation of this is provided in Fig. 4.

Given the disjunctive graph of a JSSP instance, switching a machine edge on the longest path will always result in a feasible (acyclic) schedule [20]. The choice of which edge on the longest path to flip can be determined by its rate of occurrence across all longest paths; known as its edge popularity. By flipping the most popular edge on a longest path, the likelihood of the neighbour having divergent longest paths, and hence more disparate makespan values, is greater.

However, computing all longest paths for every schedule is time consuming. We employ a different method, as follows. We switch a random machine edge and check if the resulting schedule is feasible. This can be determined by considering a sub-graph around the flipped edge in the full disjunctive graph. This allows us to efficiently check if a schedule is feasible. If it is feasible, it is considered a neighbour. If not, it is not.

An empirical analysis on real datasets showed that on average approximately 80% of edge flips resulted in a feasible schedule.

The full algorithm is shown in Algorithm 1. First, a set X of r initial schedules is generated. For each $x \in X$, a greedy local search is performed until each x is a local optimum. That is, after the greedy search, for every $x \in X$, we have that $\forall j \in N(x), L_x \leq L_j$. Let Θ be the current best solution. Simulated annealing is executed on Θ. And parallel to this, the firefly algorithm is performed using the $|X| - 1$ schedules in the set $X \setminus \Theta$.

Algorithm 1: SAFA Algorithm

input : $t_0^{(\mathrm{sa})} \leftarrow$ starting temperature for SA, $c^{(\mathrm{sa})} \leftarrow$ cooling rate for SA, $t_0^{(\mathrm{fa})} \leftarrow$ starting temperature for FA, $c^{(\mathrm{fa})} \leftarrow$ cooling rate for FA, $r \leftarrow$ number of initial solutions

output: Best schedule found

$X \leftarrow$ set of r initial schedules
for $x \in X$ **do**
    $x \leftarrow$ result of greedy local search on x until local optimum found
end
$\Theta \leftarrow \mathrm{argmin}_x L_x$
Begin SA thread with Θ
Begin Fireflies thread with $X \setminus \Theta$
Return Θ when both threads finished

Fireflies Thread

$temp^{(fa)} \leftarrow t_0^{(\mathrm{fa})}$
while $temp^{(fa)} > 1$ **do**
    for $x \in X \setminus \Theta$ **do**
        if $temp\ / t_0 > U(0,1)$ **then**
            $x \leftarrow x$ with random machine edge flipped
        else
            $x \leftarrow x$ moved towards Θ with edge flip
    end
    $temp^{(fa)} \leftarrow temp^{(fa)} \cdot (1 - c)$
    if any $L_x < L_\Theta$, replace Θ with that x
end

SA Thread

$temp^{(sa)} \leftarrow t_0^{(\mathrm{sa})}$
$y \leftarrow \Theta$
while $temp^{(sa)} > 1$ **do**
    if Θ was replaced by firefly thread, set $y \leftarrow \Theta$
    $y' \leftarrow$ random neighbour in $N(y)$
    if $L_{y'} < L_y$ **then**
        $y = y'$
    else if $temp\ / t_0 > U(0,1)$ **then**
        $y \leftarrow y'$
    $temp^{(sa)} \leftarrow temp^{(sa)} \cdot (1 - c)$
end

In a given iteration of simulated annealing, if a randomly selected neighbour solution is better than the current solution, we move to it. If not, then we move to it with some transition probability. This probability is proportional to a temperature that reduces (cools) from iteration to iteration. Early on in the SA procedure, the temperature is high and the probability of moving to worse solutions is higher compared to later in the SA procedure when the temperature is lower.

In the firefly algorithm, we also have a temperature that reduces over time. The temperature in this case, is relative to the luminescence of the firefly. We may consider the reduction in temperature equivalent to the intensification of the luminescence of Θ. With some probability proportional to the luminescence, a firefly moves towards Θ with an edge flip. Otherwise it moves to a random neighbour.

If at any point, a schedule with a makespan smaller than Θ is found, it replaces Θ, and the current schedule being considered in the SA thread is replaced by that new Θ.

Starting temperatures and cooling rates are chosen such that the number of iterations increases with input size. The starting population of the fireflies is altered such that the runtime is acceptable for the resources available.

4 Experimental Results

Table 1 shows the results of the SAFA algorithm on a selection of benchmark JSSP instances. The ID's[1] are constructed based on their origin, with benchmarks from Fisher & Thompson (ft) [8], Lawrence (la) [12], Storer, Wu and Vaccari (swv) [21], Taillard (ta) [23], Yamada and Nakano (yn) [26] and Applegate and Cook (orb) [3].

The SAFA algorithm was executed 30 times for each benchmark problem. Therefore, the SAFA AF (average found) is the average makespan found out of 30 runs. The makespan is calculated by reversing Dijkstras algorithm and finding the longest path from the source **o** to the dummy vertex *, as seen in Fig. 2.

A multitude of benchmarks have been considered for the purpose of evaluating the accuracy of the hybrid algorithm. The benchmarks have been selected based on their recognisability, size and comparability. Accordingly, results are compared to the best-found makespans, including a dissimilarity percentage between the best-found solutions of the SAFA hybrid and the known Upper Bound (UB). All computations were performed on a cloud server hosting a standard Intel dual-core processor and 7.5 gb of memory.

The SA parameter settings for the individual benchmarks are dependent on the size and complexity of the instance. In reference to Algorithm 1, the starting temperatures $t_0^{(\mathrm{sa})}$ and $t_0^{(\mathrm{fa})}$ are initiated with approximately the same value across all benchmarks. In contrast, the cooling rate parameters $c^{(\mathrm{sa})}$ and $c^{(\mathrm{fa})}$ decrease as the proportions of the benchmark increase. In other words, a lower

[1] http://jobshop.jjvh.nl.

Table 1. Results on benchmark problems. LB = lower bound. BF = best found with existing methods. SAFA BF and AF = Best found and average found respectively with SAFA method presented in this paper. % Diff = Percentage difference between the best found makespan of SAFA method compared to the best found. Note: ** = BF > LB

ID	Size	LB	BF	SAFA.BF	SAFA.AF	% Diff
ft06	6×6	55	55	55	55.00	0
ft10	10×10	930	930	937	948.63	0.7
ft20	20×5	1165	1165	1178	1181.25	1.1
la01	10×5	666	666	666	666.00	0
la10	15×5	958	958	958	958.00	0
la11	20×5	1222	1222	1222	1222.00	0
la23	15×10	1032	1032	1032	1032.00	0
la34	30×10	1721	1721	1721	1721.00	0
la35	30×10	1888	1888	1888	1888.00	0
la36	15×15	1268	1268	1306	1329.70	2.9
la37	15×15	1397	1397	1458	1485.33	4.3
swv11	50×10	2983	2983	3809	3868.10	27.6
swv13	50×10	3104	3104	3926	4013.30	26.4
swv17	50×10	2794	2794	2794	2794.00	0
swv18	50×10	2852	2852	2852	2852.00	0
ta69	100×20	3071	3071	3332	3352.33	8.4
ta71	100×20	5464	5464	6087	6230.33	11.4
ta76	100×20	5342	5342	5854	5854.00	9.5
yn02**	20×20	861	904	995	1053.80	10.0
yn03**	20×20	827	892	971	1018.00	8.8
orb01	10×10	1059	1059	1085	1089.80	2.4
orb02	10×10	888	888	895	897.30	0.7
orb03	10×10	1005	1005	1015	1037.28	0.9
orb04	10×10	1005	1005	1012	1014.50	0.6
orb05	10×10	887	887	894	895.50	0.7
orb06	10×10	1010	1010	1028	1037.75	1.7
orb07	10×10	397	397	407	407.00	2.5
orb08	10×10	899	899	928	937.50	3.2
orb09	10×10	934	934	948	953.75	1.4
orb10	10×10	944	944	957	957.00	1.3

cooling rate parameter is used for larger benchmark instances. This property allows SA to execute within a middle-ground of efficiency and efficaciousness, while enabling FA with enough iterations for fireflies to achieve their objective function - fully transitioning toward the state of the optimal schedule. This characteristic is essential for the firefly population given there is a greater likelihood of Θ being replaced when the underlying structures, and therein longest paths, become alikened to one another.

5 Conclusion

We have designed a hybrid SAFA algorithm for solving the JSSP. The algorithm has been implemented and executed on a number of different benchmarks. The experimental results are encouraging, wherein the best-found solution was reached for a number of instances and near-optimal solutions found for a wide range of others. The algorithm will be complemented by a more fine-grained analysis in the future. FA improves on the underlying SA algorithm at multiple junctures throughout its execution, with results reflecting conclusive remarks similar to other applications of FA for optimisation problems. Given the results of this paper, future work will be directed at utilising information and estimations of the backbone structure even more. In general, the combination of SA and FA works well and could further be tailored to reach best-found solutions for the larger JSSP instances as well as also being applied to solving other combinatorial optimisation problems.

References

1. Abas, M., Abbas, A., Khan, W. A.: Scheduling job shop - a case study. In: IOP Conference Series: Materials Science and Engineering, vol. 146, p. 12052. IOP Publishing (2016)
2. Adams, J., Balas, E., Zawack, D.: The shifting bottleneck procedure for job shop scheduling. Manag. Sci. **34**(3), 391–401 (1988)
3. Applegate, D., Cook, W.: A computational study of the job-shop scheduling problem. ORSA J. Comput. **3**(2), 149–156 (1991)
4. Błażewicz, J., Domschke, W., Pesch, E.: The job shop scheduling problem: conventional and new solution techniques. Eur. J. Oper. Res. **93**(1), 1–33 (1996)
5. Dorigo, M., Maniezzo, V., Colorni, A.: Positive feedback as a search strategy. Dipartimento di Elettronica, Politecnico di Milano, Italy, Technical report 91-016 (1991)
6. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: 1995 Proceedings of the Sixth International Symposium on Micro Machine and Human Science, MHS 1995, pp. 39–43. IEEE (1995)
7. Falkenauer, E., Bouffouix, S.: A genetic algorithm for job shop. In: 1991 IEEE International Conference on Robotics and Automation, Proceedings, pp. 824–829. IEEE (1991)
8. Fisher, H.: Probabilistic learning combinations of local job-shop scheduling rules. Industrial scheduling, pp. 225–251 (1963)
9. Hromkovič, J., Mömke, T., Steinhöfel, K., Widmayer, P.: Job shop scheduling with unit length tasks: bounds and algorithms. Algorithmic Oper. Res. **2**(1), 1–14 (2007)

10. Hromkovič, J., Steinhöfel, K., Widmayer, P.: Job shop scheduling with unit length tasks: bounds and algorithms. ICTCS 2001. LNCS, vol. 2202, pp. 90–106. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45446-2_6
11. Johnson, S.M.: Optimal two-and three-stage production schedules with setup times included. Naval Res. Logist. (NRL) **1**(1), 61–68 (1954)
12. Lawrence, S.: Resouce constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement). Carnegie-Mellon University, Graduate School of Industrial Administration (1984)
13. Maher, B., Albrecht, A.A., Loomes, M., Yang, X.S., Steinhöfel, K.: A firefly-inspired method for protein structure prediction in lattice models. Biomolecules **4**(1), 56–75 (2014)
14. Mömke, T.: On the power of randomization for job shop scheduling with k-units length tasks. RAIRO Theor. Inform. Appl. **43**(2), 189–207 (2009)
15. Muth, J.F., Thompson, G.L.: Industrial scheduling. Prentice-Hall, Upper Saddle River (1963)
16. Pinedo, M. L.: Scheduling: Theory, and Systems (2008)
17. Potts, C.N., Strusevich, V.A.: Fifty years of scheduling: a survey of milestones. J. Oper. Res. Soc. **60**(1), S41–S68 (2009)
18. Roy, B., Sussmann, B.: Les problemes d'ordonnancement avec contraintes disjonctives. Technical report 9 (1964)
19. Satake, T., Morikawa, K., Takahashi, K., Nakamura, N.: Simulated annealing approach for minimizing the makespan of the general job-shop. Int. J. Prod. Econ. **60**, 515–522 (1999)
20. Steinhöfel, K., Albrecht, A., Wong, C.K.: Two simulated annealing-based heuristics for the job shop scheduling problem. Eur. J. Oper. Res. **118**(3), 524–548 (1999)
21. Storer, R.H., Wu, S.D., Vaccari, R.: New search spaces for sequencing instances with application to job shop scheduling. Manag. Sci. **38**, 1495–1509 (1992)
22. Streeter, M.J., Smith, S.F.: How the landscape of random job shop scheduling instances depends on the ratio of jobs to machines. J. Artif. Intell. Res. (JAIR) **26**, 247–287 (2006)
23. Taillard, E.: Benchmarks for basic scheduling problems. Eur. J. Oper. Res. **64**(2), 278–285 (1993)
24. Vaessens, R.J.M., Aarts, E.H.L., Lenstra, J.K.: Job shop scheduling by local search. INFORMS J. Comput. **8**(3), 302–317 (1996)
25. Van Laarhoven, P.J.M., Aarts, E.H.L., Lenstra, J.K.: Job shop scheduling by simulated annealing. Oper. Res. **40**(1), 113–125 (1992)
26. Yamada, T., Nakano, R.: A genetic algorithm applicable to large-scale job-shop instances. In: Manner, Manderick (eds.) Parallel Instance Solving from Nature 2 (1992)