Minimizing Makespan using Node Based Coincidence Algorithm in the Permutation Flowshop Scheduling Problem

Abstract. This paper proposes a Node-Based Coincidence Algorithm (NB-COIN) for the permutation flowshop scheduling problems (PFSP) aimed at Makespan minimization. For almost half a century, a variety of complex algorithms have been introduced to solve the problems. Nevertheless, these algorithms will be useless if they fail to implement in practice where computational time and complexity of algorithm become an important issue of concern. NB-COIN is proved to be an effective algorithm and it is extremely fast. Based on the bench-mark data sets of Taillard, the presented algorithm provides acceptable solutions within a very short period of time. More importantly, the results generated by NB-COIN are also better than other well-known algorithms in consideration.

Keywords: Permutation flow shop, Makespan, Coincidence algorithm, Scheduling.

1 Introduction

In the highly competitive industrial market, production speed and operation cost become important factors for a manufacturing process. Producing wide variety of goods using the same production line is a key solution for many manufacturing companies. This technique is called flowshop.

In the permutation flow shop scheduling problem (PFSP), there are n jobs and m machines. All jobs have to be processed on every machine in the same order. Over the production period, all machines are ready and only one job can enter to the machines at a specific time. The pre-emption and interruption is not allowed. In order to reduce the search space, passing any jobs is prohibited in PFSP. In general, the performance is measured by two main objectives, makespan minimization or flowtime minimization. The makespan criterion is well-known to lead to rapid turn-around of jobs, uniform utilization of resources and minimization of work-in-process inventory.

The permutation flow shop scheduling problem (PFSP) has become an interesting research topic for many researchers since Johnson [1] introduced it in the 1950s. Later, the complexity of PFSP is proved to be NP-hard by Garey et al [2] and Rinnooy Kan [3]. Many heuristic optimization methods have been developed to achieve high quality solutions in a reasonable computational time such as Nawaz et al. [4], Palmer [5], Campbell et al. [6], Dannenbring[7], Taillard [8], Framinan et al. [9] and Framinan and Leisten [10]. The results given

by the most powerful heuristics, NEH, proposed by Nawaz et al. [4] are still far, at almost 7%, from the optimal value. Using only heuristics may not be capable enough to reach the optimum solution for the PFSP, many researchers developed more complex methods, metaheuristics, such as tabu search [11-14], genetic algorithms (GAs) [15-16], ant colony optimization [17-19], particle swarm optimization [20], iterated local search (ILS) [21] or the Estimation of Distribution Algorithm (EDA) [22]. Although these methods provide better results, they need to trade-off with long computational time or a lot of resources. Later, the algorithms are even enhanced by integrating two or more metaheristics, called the hybrid metaheuristics. This technique was used by G.I. Zobolas [23] and H. Liu [24] to achieve optimal solution. However a simple algorithm that can provide a reasonable solution in a short period of time is more practical in the real world. Node-Based Coincidence Algorithm (NB-COIN) is a new metaheuristic tool improved from Coincidence Algorithm (COIN) [25]. This method is proved to be an effective algorithm for flowshop scheduling problems in terms of total flow time minimization[26]. Moreover, NB-COIN is easy to implement, using very few user-defined parameters.

This paper is organized as follows: Section 2 illustrates the Permutation Flowshop Scheduling Problem. Section 3 determines the related work. The Node-Based Coincidence Algorithm (NB-COIN) is described in Section 4. Section 5 presents the computational result and the conclusion is shown in Section 6.

2 Permutation Flowshop

The makespan is the finished time of the last job in the schedule. The makespan minimization is described as $n/m/P/C_{max}$. It consists of a set J of n jobs, $J = \{j_1, \ldots, j_n\}$, and set K of m machines, $K = \{k_1, \ldots, k_m\}$. Let $t_{k,j}$ denotes as the processing times of job J on machine K and $C(k, j)$ be the completion time of job J on machine K. Thus, $C(k, j)$ can be calculated as follows:

$$C(1,1) = t_{(1,1)} \tag{1}$$

$$C(1,j) = C(1,j-1) + t_{1,j} \text{ where } j = 2, \ldots, n \tag{2}$$

$$C(k,1) = C(k-1,1) + t_{k,1}, \text{ where } k = 2, \ldots, m \tag{3}$$

$$C(k,j) = \max\{C(k,j-1), C(k-1,j)\} + t_{k,j} \tag{4}$$

3 The Related Work

In this section, a brief overview of four well-known methods for the PFSP is provided. In addition, the strength and weakness of these algorithms are pointed out. The methods include three heuristics; the NEH [4], the constructive greedy (CG)and the stochastic greedy (SG) [29], two metaheuristics; ant colony system [18] and the hybrid metaheuristic by G.I. Zobolas [23].

The performance of the NEH Heuristic (NEH) [4] has been confirmed by Park et al [27] since 1984. In addition, Turner and Booth [28], and Taillard [8],

also came to a conclusion that NEH is an efficient tool for minimizing makespan in flow shop scheduling problem. The main idea of this heuristic is that the high priority should be given to the job with more total processing time on all machines. There are 3 main steps of NEH. Step 1 order the jobs by decreasing sums of processing time, $T_j = \sum_{k=1}^{m} P_{kj}$ where P_{kj} defines as the processing time of job j in machine i. In step 2, schedule the first two jobs to minimize partial makespan. Finally, The K^{th} jobs are inserted individually at the position with the shortest makespan.

The greedy concept has been adapted to the PFSP by M. Ancau [29]. The author proposed two heuristics; the constructive greedy heuristic (CG) and the stochastic greedy heuristic (SG) and compares them against the NEH. As the result, these algorithms provide a better result in the makespan criterion, however they consume longer computational time than the well-known NEH heuristic.

The constructive heuristic algorithm (CG) generates a job's sequence using two lists called job list and optimal schedule. A job list consist of n elements $(j_1, j_2, ..., j_n)$. Firstly, a pair of jobs from the job list will be selected and arranged to find the minimum completion time passing to the optimal schedule. Then, repeat the first step, however either increase the selected elements to $k(n-k-1)$, k is the number of rounds, or pass to the optimal schedule in the relative position that minimize completion time.

In the stochastic heuristic (SG), the job list consists of n random job's elements. The first pair from the job list will be selected and finds an optimal completed time. Other jobs in the job list will be selected individually and find the best position in the optimal schedule.

K.C. Ying presented an Ant Colony System (ACS) [18] for the PFSP. This method was first introduced by Dorigo [30]. It is inspired from real ant behavior, finding shortest path using the relevant pheromones. The algorithm consists of four steps. In the first step, the method generates a set of artificial ants. Each ant employs a stochastic greedy to create a path (in the PFSF is job's sequence). The amount of pheromone is updated when the ants build a tour. Then, after all ants have completed their paths, the pheromones are modified again. The ACS is an effective algorithm that generates a solution within a small amount of time. Nevertheless, this method is sensitive to the user-defined parameters.

The hybrid metaheuristic was proposed by G.I. Zobolas[23] in 2009. This method combines different techniques and concepts from four construction heuristics and two metaheuristics to expand the solution space search and overcome the limitation of a single metaheuristic. The combination includes the heuristic proposed by Nawaz et al[4], Campbell et al. [6], Palmer's [5], Gupta's [32], and the metaheuristic algorithms such as the well-established GA [15-16] and variable neighbourhood search (VNS) [31]. In the initialization stage, the algorithm employs four well-known heuristics (NEH, Gupta, CDS, and Palmer) to generate the population. Then, the GA that adopts a special variation of the operator from Murata et al. [33] is applied for improving the population. In the third step, the VNS is used to avoid the trap of local optima. Finally, the populations are updated by replacing the old population with the new one. The results obtained

from this algorithm are achieved all optimum solution when the number of job are lower than 50.

4 Node Based Coincidence Algorithm

NB-COIN is a permutation based Estimation of Distribution Algorithm (EDA). It generates solution strings in sequences, ensuring that only valid permutations are sampled. It uses a data structure called coincidence matrix H to model substructures from absolute positions. The matrix H_{xy} represents the probability of y found in the absolute position x. The update equation of NB-COIN is

$$H_{xy}(t+1) = H_{xy}(t) + \frac{k}{n}(r_{xy}(t+1) - p_{xy}(t+1)) + \frac{k}{(n-1)^2}\left(\sum_{j=1}^{n} p_{xj}(t+1) - r_{xj}(t+1)\right) \tag{5}$$

where k denotes the learning step, n is the problem size, r_{xy} is the number of xy found in the good solutions, and p_{xj} is the number of xy found in the not-good solutions. The term $\frac{k}{(n-1)^2}\left(\sum_{j=1}^{n} p_{xj}(t+1) - r_{xj}(t+1)\right)$ represents the adjustment of all other H_{xj} where $j \neq x$ and $j \neq y$.

After each population was evaluated and ranked, two groups of candidates are selected according to their fitness values: better-group and worse-group. The better-group is selected from the top $c\%$ of the rank and is used as a reward, and H_{xy} is increased for every pair of xy found in this group. The punishment is a decrease in H_{xy} for every pair of xy found in the worse group of the bottom $c\%$ of the population rank.

The pseudo code of NB-COIN is simplified as follows:

```
Step 1: Initialize the model
Step 2: Sample the population
Step 3: Evaluate the population
Step 4: Select candidates
Step 5: Update the model
Step 6: Repeat steps 2 to 5 until terminated.
```

5 Computational Result

The proposed algorithm, Node Based Coincidence Algorithm(NB-COIN), was coded in C++ and run on MS Windows 7 using Intel Core i5 450M, 2.40GHz and 4GB of RAM. 40 instances of Taillard benchmark[8] where $n \in \{20, 50\}$ and $m \in \{5, 10, 20\}$ were selected and represented in four sets; 20×5, 20×10, 20×20 and 50×5 to determine the efficiency and performance of NB-COIN in the PFSP. Each set consist of 10 instances. Moreover, the proposed algorithm was tested according to two different criteria; computational time and performance.

5.1 Computational time

The CPU time obtained from NB-COIN were compared against the powerful metaheuristics such as ant colony system [18] and the hybrid metaheuristic proposed by G.I. Zobolas[23] in 2009 with the allocation of CPU time for 5, 15, 25 and 100 seconds to four problem sets; 20×5, 20×10, 20×20 and 50×5. In Table 1, the results obtained from all groups of instances are summarized. The computational time of NB-COIN is superior when the number of jobs is twenty, especially in the 20×5 Taillard instance. It is twice as fast as the hybrid metaheuristic and the ACS in 20×5 problem. Furthermore, the speed of hybrid metaheuristic is slower than NB-COIN by 5 and 15 seconds in 20×10 and 20×20 problems while NB-COIN is slightly slower than the ACS by 3 seconds and 9 seconds. However, the computational speed of NB-COIN decreases when the number of jobs exceeds 50.

Table 1. The computational speed

Instances	CPU time (Seconds)		
	Hybrid Metaheuristic	ACS	NB-COIN
20×5	10	11	5
20×10	20	12	15
20×20	40	16	25
50×5	25	44	100

5.2 The performance analysis

In this section, the solutions acquired from NB-COIN were tested on the Taillard benchmark against the upper bound. Although NB-COIN achieved the upper bound only in a few solutions, it is essential to mention that NB-COIN was run on the PC. It finds the high quality solution very fast while usually the upper bounds are generally generated by branch and bound techniques and run on more powerful workstations for extended time periods.

To compare the quality of solutions, the percentage gap between the makespan from our algorithm and the upper bound (UB) of Taillard. Each instance was run 5 times. To calculate the percentage gap, the equation is presented as follows:

$$Gap(\%) = \frac{C_{max} - UB}{UB} \times 100 \tag{6}$$

The results of three methods; NEH, CG and SG are adopted from the original paper proposed by Nawaz et al. [4] and M. Ancau[29] and compared against NB-COIN. Overall, we found that NB-COIN performs far better than the NEH and the constructive greedy (CG) in all problem sizes while it is slightly superior compared to the stochastic greedy (SG) in the small size problem (20×5).

Moreover, NB-COIN provides a wide variety of solutions that share the same quality.

Table 2 shows the result of Taillard's 20×5 instance. NB-COIN not only found many optimum solutions the average gap is also a lot lower than both NEH and CG. Comparing with SG algorithm, the average gap is slightly higher. However NB-COIN is better in terms of the number of good solutions.

Table 2. Performance comparison of Taillard's 20×5 instances

Instance	UB	NEH	CG	SG	NB-COIN	Gap%			
						NEH	CG	SG	NB-COIN
Ta001	1278	1286	1286	1278	1294	0.626	0.626	0	1.252
Ta002	1359	1365	1367	1366	1363	0.442	0.589	0.515	0.294
Ta003	1081	1159	1141	1097	1090	7.216	5.550	1.480	0.833
Ta004	1293	1325	1358	1306	1304	2.475	5.027	1.005	0.851
Ta005	1235	1305	1301	1244	1244	5.669	5.344	0.729	0.729
Ta006	1195	1228	1224	1210	1210	2.762	2.427	1.255	1.255
Ta007	1239	1278	1264	1251	1251	3.148	2.018	0.968	0.968
Ta008	1206	1223	1268	1206	1206	1.410	5.141	0	0
Ta009	1230	1291	1277	1253	1253	4.959	3.821	1.870	1.870
Ta010	1108	1151	1144	1117	1120	3.880	3.250	0.812	1.083
Average						3.258	3.379	0.863	0.913

The quality of solutions in the 20×10 and 20×20 problem are shown in Table 3 and Table 4. Since the performance of CG and SG algorithm for the instance where $m \in \{10, 20\}$ are not reported by M. Ancau [29]; NB-COIN is solely tested with the NEH. The results show that the average gap of NB-COIN is over three times better than the NEH in both sizes of problem.

Table 3. Performance comparison of Taillard's 20×10 instances

Instance	UB	NEH	NB-COIN	Gap%	
				NEH	NB-COIN
Ta011	1582	1680	1599	6.195	1.074
Ta012	1659	1729	1679	4.219	1.205
Ta013	1496	1557	1518	4.077	1.471
Ta014	1377	1439	1392	4.502	1.089
Ta015	1419	1502	1433	5.850	0.987
Ta016	1397	1453	1417	4.008	1.432
Ta017	1484	1562	1513	5.256	1.954
Ta018	1538	1609	1575	4.616	2.406
Ta019	1593	1647	1608	3.390	0.942
Ta020	1591	1653	1617	3.897	1.634
Average				4.601	1.419

Table 4. Performance comparison of Taillard's 20×20 instances

Instance	UB	NEH	NB-COIN	Gap%	
				NEH	NB-COIN
Ta021	2297	2410	2323	4.919	1.132
Ta022	2099	2150	2119	2.430	0.953
Ta023	2326	2411	2349	3.654	0.989
Ta024	2223	2262	2242	1.754	0.855
Ta025	2291	2397	2314	4.627	1.004
Ta026	2226	2349	2243	5.526	0.764
Ta027	2273	2362	2300	3.915	1.188
Ta028	2200	2249	2235	2.227	1.591
Ta029	2237	2320	2276	3.710	1.743
Ta030	2178	2277	2200	4.545	1.010
Average				3.731	1.123

As seen in Table 5, NB-COIN performs very well in Taillard's 50×5 instance. It is clear that the SG provides slightly better results in this size of problem. However, NB-COIN found more optimum solutions than SG and has a lower average gap than both the NEH and CG algorithm. In addition, in each instance, although the average gap of SG algorithm is slightly lower than NB-COIN, the SG consumes more CPU time, at almost double.

Table 5. Performance comparison of Taillard's 50×5 instances

Instance	UB	NEH	CG	SG	NB-COIN	Gap%			
						NEH	CG	SG	NB-COIN
Ta031	2724	2733	2761	2724	2724	0.330	1.358	0	0
Ta032	2834	2843	2889	2848	2848	0.317	1.941	0.494	0.494
Ta033	2621	2640	2674	2622	2640	0.725	2.022	0.038	0.725
Ta034	2751	2782	2782	2782	2771	1.127	1.127	1.127	0.727
Ta035	2863	2868	2908	2863	2863	0.175	1.572	0	0
Ta036	2829	2850	2863	2840	2835	0.742	1.202	0.389	0.212
Ta037	2725	2758	2781	2732	2739	1.211	2.055	0.257	0.514
Ta038	2683	2721	2780	2701	2704	1.416	3.615	0.671	0.783
Ta039	2552	2576	2595	2562	2565	0.940	1.685	0.392	0.510
Ta040	2782	2790	2787	2784	2782	0.287	0.180	0.072	0
Average						0.727	1.676	0.343	0.396

Overall, the quality of solutions as measured by gap averaging over all test instances (Table 2-5) is 0.96% from the upper bound.

6 Conclusion

In this paper, we present a new method to solve the permutation flowshop scheduling problem called the Node Based Coincidence Algorithm (NB-COIN). This algorithm makes use of positive and negative knowledge to rapidly improve the solution. The proposed method was tested on a set of 40 Taillard instances. The experiment shows that the solution of NB-COIN is very close to the optimal value, at only 0.96% from the upper bound on average. Moreover, the proposed algorithm not only provides acceptable results very fast, it has few user-defined parameters. Hence, NB-COIN is a highly appropriate method that is easy to apply to real world situations where lower computational time and higher quality solutions are preferred.

References

1. Johnson, S.M.: Optimal twoand threestage production schedules with setup times included, Naval research logistics quarterly, 1954, 1, (1), pp. 61-68
2. Garey, M.R., Johnson, D.S., and Sethi, R.: The complexity of flowshop and jobshop scheduling, Mathematics of operations research, 1976, 1, (2), pp. 117-129
3. Lenstra, J.K., Kan, A.R., and Brucker, P.: Complexity of machine scheduling problems, Annals of discrete mathematics, 1977, 1, pp. 343-362
4. Nawaz, M., Enscore, E.E., and Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, Omega, 1983, 11, (1), pp. 91-95
5. Palmer, D.: Sequencing jobs through a multi-stage process in the minimum total time–a quick method of obtaining a near optimum, OR, 1965, pp. 101-107
6. Campbell, H.G., Dudek, R.A., and Smith, M.L.: A heuristic algorithm for the n job, m machine sequencing problem, Management science, 1970, 16, (10), pp. B-630-B-637
7. Dannenbring, D.G.: An evaluation of flow shop sequencing heuristics, Management science, 1977, 23, (11), pp. 1174-1182
8. Taillard, E.: Some efficient heuristic methods for the flow shop sequencing problem, European journal of Operational research, 1990, 47, (1), pp. 65-74
9. Framinan, J.M., Leisten, R., and Ruiz-Usano, R.: Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation, European Journal of Operational Research, 2002, 141, (3), pp. 559-569
10. Framinan, J., and Leisten, R.: An efficient constructive heuristic for flowtime minimisation in permutation flow shops, Omega, 2003, 31, (4), pp. 311-317
11. Grabowski, J., and Wodecki, M.: A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion, Computers Operations Research, 2004, 31, (11), pp. 1891-1909
12. Nowicki, E., and Smutnicki, C.: A fast tabu search algorithm for the permutation flow-shop problem, European Journal of Operational Research, 1996, 91, (1), pp. 160-175

13. Reeves, C.R.: Improving the efficiency of tabu search for machine sequencing problems, Journal of the Operational Research Society, 1993, pp. 375-382
14. Watson, J.-P., Barbulescu, L., Whitley, L.D., and Howe, A.E.: Contrasting structured and random permutation flow-shop scheduling problems: search-space topology and algorithm performance, INFORMS Journal on Computing, 2002, 14, (2), pp. 98-123
15. Reeves, C.R.: A genetic algorithm for flowshop sequencing, Computers operations research, 1995, 22, (1), pp. 5-13
16. Reeves, C.R., and Yamada, T.: Genetic algorithms, path relinking, and the flowshop sequencing problem, Evolutionary computation, 1998, 6, (1), pp. 45-60
17. Stützle, T.: An ant approach to the flow shop problem, in Editor : Book An ant approach to the flow shop problem (1998, edn.), pp. 1560-1564
18. Rajendran, C., and Ziegler, H.: Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs, European Journal of Operational Research, 2004, 155, (2), pp. 426-438
19. Ahmadizar, F., Barzinpour, F., and Arkat, J.: Solving permutation flow shop sequencing using ant colony optimization, in Editor : Book Solving permutation flow shop sequencing using ant colony optimization (IEEE, 2007, edn.), pp. 753-757
20. Tasgetiren, M.F., Liang, Y.-C., Sevkli, M., and Gencyilmaz, G.: A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem, European Journal of Operational Research, 2007, 177, (3), pp. 1930-1947
21. Stützle, T.: Applying iterated local search to the permutation flow shop problem, FG Intellektik, TU Darmstadt, Darmstadt, Germany, 1998
22. Jarboui, B., Eddaly, M., and Siarry, P.: An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems, Computers Operations Research, 2009, 36, (9), pp. 2638-2646
23. Zobolas, G., Tarantilis, C.D., and Ioannou, G.: Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm, Computers Operations Research, 2009, 36, (4), pp. 1249-1267
24. Liu, H., Gao, L., and Pan, Q.: A hybrid particle swarm optimization with estimation of distribution algorithm for solving permutation flowshop scheduling problem, Expert Systems with Applications, 2011, 38, (4), pp. 4348-4360
25. Wattanapornprom, W., Olanviwitchai, P., Chutima, P., and Chongstitvatana, P.: Multi-objective Combinatorial Optimisation with Coincidence algorithm, in Editor : Book Multi-objective Combinatorial Optimisation with Coincidence algorithm (2009, edn.), pp. 1675-1682
26. Srimongkolkul, O., and Chongstitvatana, P.: Application of Node Based Coincidence algorithm for flow shop scheduling problems, in Editor : Book Application of Node Based Coincidence algorithm for flow shop scheduling problems (2013, edn.), pp. 49-52
27. Park, Y.B., Pegden, C.D., and Enscore, E.E.: A survey and evaluation of static flowshop scheduling heuristics, The International Journal of Production Research, 1984, 22, (1), pp. 127-141

28. Turner, S., and Booth, D.: Comparison of heuristics for flow shop sequencing, Omega, 1987, 15, (1), pp. 75-78
29. Ancu, M.: On Solving Flowshop Scheduling Problems, Proceedings of the Romanian Academy. Series A, 2012, 13, (1), pp. 71-79
30. Dorigo, M.: Optimization, learning and natural algorithm, Politecnico di Milano, 1992
31. Hansen, P., and Mladenovi, N.: Variable neighborhood search: Principles and applications, European journal of operational research, 2001, 130, (3), pp. 449-467
32. Gupta, J.N.: A functional heuristic algorithm for the flowshop scheduling problem, Operational Research Quarterly, 1971, pp. 39-47
33. Murata, T., Ishibuchi, H., and Tanaka, H.: Genetic algorithms for flowshop scheduling problems, Computers Industrial Engineering, 1996, 30, (4), pp. 1061-1071