IOCA 2021

MDPI

Proceedings

Parallel WSAR for Solving Permutation Flow Shop Scheduling Problem †

Adil Baykasoğlu [1], Mümin Emre Şenol [2,*]

[1] Dokuz Eylül University, Faculty of Engineering, Department of Industrial Engineering, Izmir, TURKEY; adil.baykasoglu@deu.edu.tr
[2] Dokuz Eylül University, Faculty of Engineering, Department of Industrial Engineering, Izmir, TURKEY; emre.senol@deu.edu.tr
* Correspondence: emre.senol@deu.edu.tr; Tel.: +90 232 301 76 21

† Presented at the title, place, and date.

Abstract: This study presents a coalition-based parallel metaheuristic algorithm for solving Permutation Flow Shop Scheduling Problem (PFSP). The novel approach incorporates five different single-solution based metaheuristic algorithm (SSBMA) (Simulated Annealing Algorithm, Random Search Algorithm, Great Deluge Algorithm, Threshold Accepting Algorithm and Greedy Search Algorithm) and a population based algorithm (Weighted Superposition Attraction-Repulsion Algorithm) (WSAR). While SSBMAs are responsible for exploring the search space, WSAR serves as a controller that handles the coalition process. SSBMAs perform their search simultaneously through MATLAB parallel programming tool. The proposed approach tested on PFSP against the state of the art algorithms in the literature. Moreover, the algorithm is also tested against its constituents (SSBMAS and WSAR) and its serial version. Non-parametric statistical tests are organized to compare the performance of the proposed approach statistically with the state of the art algorithms, its constituents and its serial version. The statistical results prove the effectiveness of the proposed approach.

Keywords: Parallel computing; Coalition; Permutation Flow Shop Scheduling Problem

Citation: Baykasoglu A.; Senol, M.E. Parallel WSAR for solving Permutation Flow Shop Scheduling Problem. *Proceedings* **2021**, *68*, x. https://doi.org/10.3390/xxxxx

Published: date

1. Introduction

Optimization is finding the solution that gives the best result in the solution space of a problem. In other words, it is to achieve the best solutions under the given conditions. Today, different optimization algorithms are used to solve many optimization problems [1-4]. These algorithms can be classified in two groups as exact algorithms and approximate algorithms. Exact algorithms search the entire search space and try every possible alternative solution. Even if they provide the optimal solution, they need long runtime, especially as the size of the problem grows. On the other hand, approximate algorithms perform their solution space search through some logical operators. Although they do not guarantee optimal solution, they provide near optimal solutions in reasonable time. Through this superiority, most of the researchers prefer approximate algorithms in optimization problem solving.

Approximate algorithms are classified into two groups as heuristic and metaheuristic algorithms. While a heuristic algorithm's structure is problem-specific, a metaheuristic algorithm's structure is generic, allowing it to be applied to any optimization problem. Metaheuristic algorithms are more flexible than heuristic algorithms in that they can handle any problem. They can also provide better solutions to optimization problems than heuristic algorithms. Metaheuristic algorithms, on the other hand, may have drawbacks such as early convergence and poor speed, and a metaheuristic algorithm may be superior to other metaheuristic algorithms.

The No Free Lunch Theorem [5] must also be mentioned at this point in order to underline the logic for integrating diverse search techniques within the framework of creating successful optimization methods. According to this theorem, no optimization method beats all remaining solution processes for all optimization problems, and there is no statistical difference between the performances of different metaheuristics when all optimization problems are solved [6]. It is a result that implies that the computing cost of finding a solution for optimization problems is the same for any solution technique. This theorem can be a base point to combine various metaheuristic algorithms to tackle optimization problems more effectively. It will take substantial time to combine the various metaheuristic algorithms and run them sequentially [7]. Most of the metaheuristic algorithms are designed to run sequentially, parallel execution of metaheuristic algorithms can increase solution quality while shortening run time [8][9].

This research is the outcome of an attempt to combine several metaheuristics in order to reveal a high level of synergy and, as a result, deliver sufficient performance while solving optimization problems.

This paper provides a new framework for addressing Permutation Flow Shop Scheduling Problem (PFSP) based on a coalition of diverse metaheuristics in a parallel computing environment. To implement the multiple metaheuristic algorithms in parallel, a new optimization system combining different single solution based metaheuristic algorithms(SSBMA) (Simulated Annealing Algorithm (SA), Random Search Algorithm(RS), Great Deluge Algorithm(GD), Threshold Accepting Algorithm (TA) and Greedy Search Algorithm (GS)) and a controller (Weighted Superposition Attraction algorithm) is designed.

The remaining of the paper is organized as follows. In Section 2, parallel computing is explained and in Section 3, the proposed optimization approach (p-WSAR) is introduced. In Section 4, PFSP is presented and experimental results are reported. Finally, concluding remarks are presented in Section 5.

2. Parallel Computing

Parallel computing is a type of computing architecture in which many processors execute or process an application or computation simultaneously. Parallel computing helps us do large computations by dividing the workload among multiple processors, all working on at the same time. Most supercomputers use parallel computing principles to work. Parallel computing is also known as parallel processing. For this to happen, we need to properly empower resources to execute concurrently. Parallel computing can reduce solution time, increase energy efficiency in our application, and allow us to tackle bigger problems. It is a computational technique developed to solve complex problems faster and more efficiently [10] [11].

3. p-WSAR Algorithm

The p-WSAR algorithm is introduced in this section. p-WSAR is comprised of five SSBMAs namely, Random Search (RS) [12], Threshold Accepting (TA) [13], Great Deluge [14], Simulated Annealing (SA) [15], Greedy Search (GS) [16] and a controller WSAR [17]. p-WSAR mainly has three stages namely search stage, information sharing stage and reproduction stage. In the search stage, all of the SSBMAs explores the solution space in parallel. After exploring the solution space, they share their findings with other SSBMAs through WSAR algorithm superposition principle. One can see the details of the superposition principle in the following study [17]. Then, all SSBMAs moves through their next positions. In the last stage, SSBMAs' parameters reproduced. This iterative process lasts until the termination criteria is met. Notations of p-WSAR algorithm is given below.
The main stages of the WSAR algorithm and flow chart of the algorithm are depicted in Figure 1 and Figure 2 respectively.

```
PROCEDURE p-WSAR
Randomly generate initial solutions
Match randomly generated initial solutions with randomly selected SSBMAs
WHILE iteration<MaxIter
    WHILE termination conditions for SSBMAs are not met
        Run SSBMAs in parallel
    END WHILE
    Sort the solutions returned by SSBMAs according to their fitness values
    Determine attractive and repulsive superpositions
    Calculate attractive and repulsive superpositions' fitness
        PARFOR each solution returned by SSBMAs   //parallel FOR
            IF solution_fitness < Superposition fitness
                Randomly move solutions
            ELSE
                Move towards superposition
            END IF
            Randomly generate a new set of parameters for SSBMAs
            Randomly match solutions with SSBMAs
        END FOR
END WHILE
END PROCEDURE
```

Figure 1. Main steps of p-WSAR

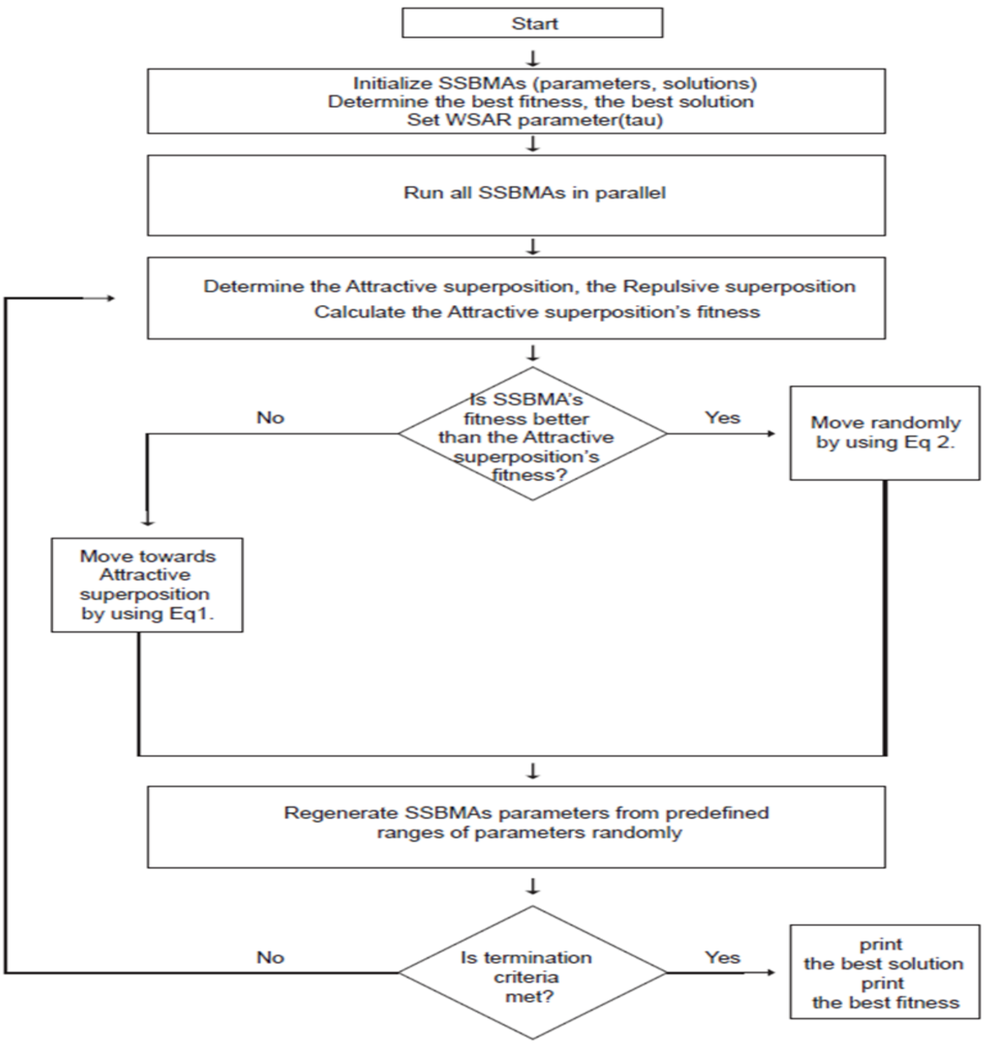

Figure 2. The flow chart of the p-WSAR algorithm

4. Permutation Flow Shop Scheduling Problem and Experimental Results

In this section, firstly PFSP is introduced and then, experimental results are given

4.1 Permutation Flow Shop Scheduling Problem (PFSP)

The PFSP has a set of m machines and a group of n jobs. Every job is made up of m operations that must be accomplished on several machines. For each of the n jobs, the machine ordering for the process sequence is the same. Each machine may only conduct one operation at a time, and all jobs are completed sequentially according to a permutation schedule. It is assumed that no machine problems would occur during the manufacturing stage, thus all of the machines will be ready to process activities. Operation preemption is also disallowed. The goal is to design a schedule that reduces the total job completion time (makespan) while adhering to the preceding assumptions.

A permutation type n-dimensional real-number vector can be utilized in the PFSP to determine the job process sequence. After identifying the job order, the makespan can be calculated using the "completion time matrix approach," which Onwubolu and Davendra proposed [18].

4.2 Experimental Results

The p-WSAR's performance in PFSP is evaluated using the Taillard [19] benchmark instances, which are divided into 12 groups of problems. 5 of these problems are selected to test p-WSAR's performance against some state of the art algorithms and WSAR. These problems' size (PS: (J*M) and well known solutions (WKS) is given in Table 1. The best, the worst and the average performance of 30 runs of each algorithm is recorded. In all of the instances, p-WSAR is able to find better solutions than other algorithms.

Table 1. Comparison of p-WSAR with some state of the art algorithms and WSAR

Problems	Algorithm	TLBO[20]	HPSO[21]	NPSO[22]	WSAR	p-WSAR
ta001	Best	1278	1278	1278	1278	1278
PS:(20*5)	Worst	1297	1278	1297	1297	1278
WKS:1278	Average	1287.2	1278	1279.9	1278.6	1278
ta011	Best	1586	1582	1582	1586	1582
PS:(20*10)	Worst	1618	1596	1639	1618	1582
WKS:1582	Average	1606	1587.3	1605.8	1592.2	1582
ta031	Best	2724	2724	2724	2724	2724
PS:(50*5)	Worst	2741	2724	2729	2729	2724
WKS:2724	Average	2729.4	2724	2725	2724.6	2724
ta051	Best	3986	3923	3938	3969	3902
PS:(50*20)	Worst	4095	3963	3989	4063	3923
WKS:3771	Average	4029.7	3944.6	3964.3	4015.9	3916
ta061	Best	5493	5493	5493	5493	5493
PS:(100*5)	Worst	5527	5493	5495	5495	5493
WKS:5493	Average	5499.4	5493	5493.2	5493.2	5493

In addition, the performance of p-WSAR is statistically compared with the other algorithms through nonparametric statistical tests by using average values. Table 2 indicates that (based on the Friedman test results) p-WSAR surpasses the other algorithms. Furthermore, according to the Wilcoxon signed-rank test, the difference between p-WSAR and HPSO is found negligible as the $p > 0.1$. Besides p-WSAR is performed slightly better than TLBO, NPSO, WSAR as $p < 0.1$.

Table 2. Non-parametric test results on Taillard Instances

Friedman test average rankings			Wilcoxon signed-rank test between p-WSAR and state of the art algorithms	
Algorithms	Sum of Ranks		p-WSAR vs.	p-value
TLBO	5.0	(5)	TLBO	0.0625
HPSO	1.7	(2)	HPSO	0.5
NPSO	3.7	(4)	NPSO	0.0625
WSAR	3.3	(3)	WSAR	0.0625
p-WSAR	1.3	(1)		

Another computational study is organized to test the performance of p-WSAR with its constituents (SSBMAs) in terms of solution quality. The results are presented in Table 3, and Table 4. According to the computational results, p-WSAR' performance is far beyond its constituents (SSBMAs). Besides, in respect of the non-parametric statistical tests, p-WSAR is able to produce more effective results than its constituents. Also, there is statistically significant difference between the performance of the p-WSAR and its constituents since p-value is < 0.1.

Table 3. Comparison of p-WSAR with SSBMAs

Problems	Algorithm	SA	RS	GD	TA	GS	p-WSAR
ta001	Best	1286	1294	1278	1278	1284	1278
PS:(20*5)	Worst	1297	1302	1297	1284	1292	1278
WKS:1278	Average	1292.2	1296.5	1279.9	1280.6	1287.8	1278
ta011	Best	1606	1616	1596	1592	1608	1582
PS:(20*10)	Worst	1620	1650	1616	1618	1642	1582
WKS:1582	Average	1610	1632.4	1610.7	1608	1624	1582
ta031	Best	2804	2942	2806	2864	2916	2724
PS:(50*5)	Worst	2908	3026	2846	2938	3002	2724
WKS:2724	Average	2856	2978	2824.6	2886	2984	2724
ta051	Best	4206	4807	4402	4622	4424	3902
PS:(50*20)	Worst	4240	6240	4803	5162	6024	3923
WKS:3771	Average	4222.4	5465.8	4627	4838.6	5146	3916
ta061	Best	6122	8640	6248	6125	7426	5493
PS:(100*5)	Worst	6378	9026	6414	6642	8424	5493
WKS:5493	Average	6564.3	8924.7	6344.9	6348.4	8012.6	5493

Table 4. Non-parametric test results on Taillard Instances p-WSAR vs. SSBMAs

Friedman test average rankings			Wilcoxon signed-rank test between p-WSAR and state of the art algorithms	
Algorithms	Sum of Ranks		p-WSAR vs.	p-value
SA	3.4	(4)	SA	0.0625
RS	5.8	(6)	RS	0.0625
GD	2.6	(2)	GD	0.0625
TA	3.2	(3)	TA	0.0625
GS	5.0	(5)	GS	0.0625
p-WSAR	1.0	(1)		

5.Conslusion

In this research, multiple metaheuristic algorithms are combined to build a coalition for tackling PFSP. The suggested methodology uses WSAR as the controller to run multiple single solution based metaheuristic algorithms (SSBMAs) in parallel. The suggested method is put to the test on some of the Taillard instances. According to the results, the proposed approach is capable of finding the best solutions. Furthermore, the proposed approach surpasses its constituents. The proposed approach's motivation is supported by the computational results. Applying the proposed approach to the other type of problems is planned as a future research.

References

1. Precup, R.-E., David, R.-C., Roman, R.-C., Petriu, E. M., & Szedlak-Stinean, A.-I. (2021).Slime Mould Algorithm-Based Tuning of Cost-Effective Fuzzy Controllers for Servo Systems. *International Journal of Computational Intelligence Systems*, *14*(1), 1042–1052.
2. Ang, K. M., Lim, W. H., Isa, N. A. M., Tiang, S. S., & Wong, C. H. (2020). A constrained multi-swarm particle swarm optimization without velocity for constrained optimization problems. *Expert Systems with Applications*, *140*, 112882.
3. Mirjalili, S., Gandomi, A. H., Mirjalili, S. Z., Saremi, S., Faris, H., & Mirjalili, S. M. (2017).Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems. *Advances in Engineering Software*, 114, 163–191.
4. Baykasoğlu, A., Dudaklı, N., Subulan, K., & Taşan, A. S. (2021). An integrated fleet planning model with empty vehicle repositioning for an intermodal transportation system. *Operational Research*, 1-36
5. Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82. https://doi.org/10.1109/4235.585893
6. Malek, R. "Collaboration of metaheuristic algorithms through a multi-agent system." *International Conference on Industrial Applications of Holonic and Multi-Agent Systems*. Springer, Berlin, Heidelberg, 2009.
7. Baykasoğlu, A., Hamzadayi, A., & Akpinar, S. (2019). Single Seekers Society (SSS): Bringing together heuristic optimization algorithms for solving complex problems. *Knowledge-Based Systems*, *165*, 53-76.
8. Alba, E. (2005). Parallel Metaheuristics: A New Class of Algorithms. In Parallel Metaheuristics: A New Class of Algorithms. https://doi.org/10.1002/0471739383
9. Alba, E., & Troya, J. M. (2002). Improving flexibility and efficiency by adding parallelism to genetic algorithms. *Statistics and Computing*, *12(2)*, 91–114. https://doi.org/10.1023/A:1014803900897
10. Almasi, G. S., & Gottlieb, A. (1994). *Highly parallel computing*. Benjamin-Cummings Publishing Co., Inc..
11. Kohli, R., & Krishnamurti, R. (1989). Optimal product design using conjoint analysis: Computational complexity and algorithms. *European Journal of Operational Research*, *40*(2), 186-195.
12. Rogers, D. (1972). Random Search and Insect Population Models. *The Journal of Animal Ecology*, *41(2)*, 369. https://doi.org/10.2307/3474
13. Dueck, G., & Scheuer, T. (1990). Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, *90(1)*, 161–175. https://doi.org/10.1016/0021-9991(90)90201-
14. Dueck, G. (1993). New optimization heuristics; The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, *104(1)*, 86–92. https://doi.org/10.1006/jcph.1993.1010
15. Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220(4598)*, 671–680. https://doi.org/10.1126/science.220.4598.671
16. Feo, T. A., & Resende, M. G. C. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, *6(2)*, 109–133. https://doi.org/10.1007/BF01096763
17. Baykasoğlu, A. (2020). Optimising cutting conditions for minimising cutting time in multi-pass milling via weighted superposition attraction-repulsion (WSAR) algorithm. *International Journal of Production Research*, 2020. https://doi.org/10.1080/00207543.2020.1767313
18. Onwubolu, G., Davendra, D., 2006. Scheduling flow shops using differential evolution algorithm. *Eur. J. Oper. Res. 171*, 674–692. https://doi.org/10.1016/j.ejor.2004.08.043
19. Taillard, E., 1990. Some efficient heuristic methods for the flow shop sequencing problem. *Eur. J. Oper. Res. 47*, 65–74. https://doi.org/10.1016/0377-2217(90)90090-X
20. Baykasoğlu, A., Hamzadayi, A., Köse, S.Y., 2014. Testing the performance of teaching-learning based optimization (TLBO) algorithm on combinatorial problems: Flow shop and job shop scheduling cases. *Inf. Sci. (Ny). 276*, 204–218. https://doi.org/10.1016/j.ins.2014.02.056
21. Lin, S.Y., Horng, S.J., Kao, T.W., Huang, D.K., Fahn, C.S., Lai, J.L., Chen, R.J., Kuo, I.H., 2010. An efficient bi-objective personnel assignment algorithm based on a hybrid particle swarm optimization model. *Expert Syst. Appl. 37*, 7825–7830. https://doi.org/10.1016/j.eswa.2010.04.056

22. Lian, Z., Gu, X., Jiao, B., 2008. A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan. *Chaos, Solitons and Fractals 35*, 851–861. https://doi.org/10.1016/j.chaos.2006.05.082