A Modern Approach to the Permutation Flowshop Problem (PFSP)

Ilias Mpourdakos[0009−0002−2621−5264]

Athens University of Economics and Business, Greece, https://aueb.gr/

Abstract. The *Permutation Flowshop Problem* (PFSP) describes any context where a set of jobs, J, are issued to be processed by a sequence of M machines where every j-m mapping, has its own, *predefined*, processing time. Each job has to pass through every machine and the machines have a predefined sequence. Every different sequence, s, of J is mapped to a value called the makespan, which is the completion time of the $j \in J$ in the last position of s. The problematic nature of the abovementioned comes when there is a need for the jobs to complete, given the *minimum* amount of time, meaning the minimization of makespan. While many variants of the Flowshop problem have been seen in the literature, each describing a context under different constraints, the *permutation* element of this problem has the characteristic that only **one** job permutation will exist as the final solution, and that will be held for **every machine**.

1 Reason of Being

This brief report aims to describe, model and finally propose an implementation that can be used to approach the PFSP using literature methods. The topics appear as follows: In section 2, with the goal of describing the problem properly, an introduction to the methods used is being made, as well as the reason for their selection. In section 3, a final implementation is being proposed, using some of the methods of section 2. Finally, section 4 mentions the computational results of the implementation in 3, as well as the process of parameter setting of the algorithms.

2 Methodology

In the field of Operations Research, many methods have been proposed by various researchers (some of them referenced in this report) to solve the PFSP. While their methodology and strategies may differ, all of them have to make a choice regarding the form of their proposed solution; that being an *exact* methodology or a *heuristic* one. In the instance that an exact method will be used to approach the problem, such as an MILP (Mixed Integer Linear Programming) model, while the solution will give an objective equal to the global optima, there always exists a trade-off between solution quality and computational time. In

the case of the PFSP, with its solution space being $n!$, where n is the number of jobs scheduled for processing by the machines, these models will require a great amount of time to produce a solution, even on small instances. On the other side, by using heuristic and meta-heuristic designs to solve the same problem, a final solution can be reached, not far from the global optima (if using the right methods), that can be produced in a fraction of the time of the aforementioned models. That's why the proposed implementation in this report, uses heuristic and meta-heuristic schemas in order to reach a solution. First a solution is constructed using a greedy method, and then an iterative improvement algorithm tries to better the already found solution.

3 Proposed Implementation

The proposed implementation stems from the works of Nawaz-Enscore-Ham [1] and Glover [2]. To be more precise, the implementation uses a starting solution that is constructed by the NEH algorithm [1] and then, using a Tabu Search [2] exploration technique, the solution gets reformed while trying to escape local optima. The Tabu settings that are being used, specify the rule of full-solution records in the Tabu List, meaning that at every move, the whole solution is being stored in the short-term memory of the algorithm, and not just the features of said solution. The selection of this approach stems from the critical factor to completely remove solution cycling in the neighborhood exploration. The tuning results of the algorithm are described in detail in section 4.

Algorithm 1 PERMUTATION PROCEDURE

Input: J, a set of jobs. M, a sequence of machines.
Data: p_{jm}, the processing times of jobs on machines.
Output: *solution*, the permutation of J, that has the *minimum* makespan.

1: **Let** $memorySize = 11$;
2: **Let** $iterations = 10$;
3: $startSolution$ = **NEH**(J, M, p_{jm});
4: $improvedSolution$ = **TabuSearch**($startSolution$, $memorySize$, $iterations$);
5: **return** $improvedSolution$;

3.1 Permutation Algorithm

The execution of Algorithm 1 requires an input of jobs and machines, as well as the processing time of every job-machine pair. In line [1] and [2], the parameters of the neighborhood exploration are being set. That is, the $memorySize$, that denotes the maximum number of solutions that can be disregarded while searching, and $iterations$, that specifies how many moves, meaning iterations, should the exploration consider before terminating. In line [3], a starting solution is being constructed and in line [4] that solution gets improved, before returning it and terminating the procedure, in line [5].

4 Computational Evidence

There still exists a mystery regarding the move type that will be used in the exploration procedure. In the process of algorithm tuning, 2 operators were considered. Them being the 1-1 exchange operator (or "swap" operator), and the 1-0 exchange operator (or "relocation" operator). In order to evaluate their performance in different exploration environments (different values of memory size and iterations before terminating), to find the optimal settings, many experiments were run and the results can be seen in Table 1. The instance that was used for the experiments was Taillard's [3] TA001 proposal, that being a 20 job, 5 machine problem.

Table 1. Detailed Experimental Results

Method	**Cost**	**Time** (s)	**Memory Size**	**Iterations**
NEH	1310	0.01425	-	-
TABU 1-0	1278	20.68	11	10
TABU 1-1	1297	14.55	5	10
BASIC 1-0	1297	3.89	-	-
BASIC 1-1	1297	1.91	-	-

the parameters that were used on the experiments are as follows:
Tabu List Memory Size: 5, 6, 7, 8, 9, 10, 11, 12 elements.
Maximum Number of Iterations: 10, 15, 20.

Each method evaluation on a unique parameter combination setting was taken on an average of 10.

It is evident that the Relocation operator wields greater results, reaching a minimum cost of 1278, in just 20.7 seconds of execution time. The Swap operator on the other hands, only manages to reduce the cost to 1297, despite the iterations number or memory size tested. Table 1 depicts the minimum time that the move types needed to reach said objective, so it is obvious that since we have no improvements on the cost with the Swap operator, regardless of the parameters tested, that time would appear when the maximum iterations of the search are 10 (that being the minimum number of iterations of the tests). To cover a greater range of results, the data set was also tested with a basic schema of a local search algorithm, using the same operators. By using a basic schema instead of a Tabu search, it appears that the best solution that can be reached has an objective of 1297, even when using the relocation move type.

All experiments have been performed on a machine with Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz 2.40 GHz and 8GB RAM, running Windows 10 operating system. The entirety of the algorithms were written in C++ using the GCC v. 12.1 compiler distribution.

References

1. Muhammad Nawaz, E Emory Enscore, Inyong Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, Omega, Volume 11, Issue 1, 1983, Pages 91-95, ISSN 0305-0483. doi: https://doi.org/10.1016/0305-0483(83)90088-9.
2. Fred Glover, (1989) Tabu Search—Part I. ORSA Journal on Computing 1(3):190-206. doi: https://doi.org/10.1287/ijoc.1.3.190
3. E. Taillard, Benchmarks for basic scheduling problems, European Journal of Operational Research, Volume 64, Issue 2, 1993, Pages 278-285, ISSN 0377-2217. doi: https://doi.org/10.1016/0377-2217(93)90182-M.