A HyFlex Module for the Permutation Flow Shop Problem

José Antonio Vázquez-Rodríguez, Gabriela Ochoa, Tim Curtois
Matthew Hyde
Automated Scheduling, Optimisation and Planning (ASAP) Group,
School of Computer Science, University of Nottingham, Jubilee Campus, Wollaton Road,
Nottingham. NG8 1BB. UK

1 Problem Formulation

The permutation flow shop problem requires to find the order in which n jobs are to be processed in m consecutive machines. The jobs are processed in the order machine 1, machine $2, \ldots,$ machine m. Machines can only process one job at a time and jobs can be processed by only one machine at a time. No job can jump over any other job, meaning that the order in which jobs are processed in machine 1 is maintained throughout the system. Moreover, no machine is allowed to remain idle when a job is ready for processing. All jobs and machines are available at time 0. Each job i requires a processing time on machine j denoted by p_{ij}.

Given a permutation $\pi = \pi(1), \ldots, \pi(n)$, where $\pi(q)$ is the index of the job assigned in the q-th place, a unique schedule is obtained by calculating the starting and completion time of each job on each machine. The starting time $start_{\pi(q),j}$ of the q-th job on machine j is calculated as:

$$start_{\pi(q),j} = \max\{start_{\pi(q),j-1}, start_{\pi(q-1),j}\},$$

with

$$start_{\pi(0),j} = 0 \quad \text{and} \quad start_{\pi(q),0} = 0,$$

and its completion time is calculated as:

$$C_{\pi(q),j} = start_{\pi(q)} + p_{\pi(q),j}.$$

Given a schedule, let C_i be the time when job i finishes its processing on machine m. The permutation flow shop problem requires to find the processing order of n jobs in such a way that the resultant schedule minimises the completion time of the last job to exit the shop, i.e. minimises $\max_i C_i$.

The problem domain module described in this report offers a set of operators to initialise and modify solutions which are commonly found in effective meta-heuristics and a set of benchmarks instances (due to [10]) that are readily available. These are explained in detail next.

2 Solution initialisation

The method `initialiseSolution`(b) creates a new solution and places it in slot b in `memory`. The solution is created with a randomised version of the well established NEH algorithm [1]. The NEH procedure has been used as an important component of many effective meta-heuristics for the permutation flow shop problem [2, 3, 4, 5]. It works as follows. First a random permutation of the jobs is generated. Second, a schedule is constructed from scratch by assigning the first job in the permutation to an empty schedule; the second job is then assigned to places 1 and 2 and fixed where the partial schedule has the smallest makespan; the third job is assigned to places 1, 2 and 3 and fixed to the place where the partial schedule has the smallest makespan, and so on.

3 Low level heuristics

A total of 15 low level heuristics $h_1, \ldots, h_{15}$ were implemented. These are described next. Please bear in mind that all of the following heuristics act on an initial solution s (permutation of job indices) and produce a new solution s'.

3.1 Mutational heuristics

h_1: Reinserts a randomly selected job into a randomly selected position in he permutation, shifting the rest of the jobs as required.

h_2: Swaps two randomly selected jobs in the permutation.

h_3: Shuffles randomly the entire permutation.

h_4: Creates a new solution using NEH and using the current permutation to rank the jobs.

h_5: Shuffles k randomly selected elements in the permutation, where $k = 2 + \lfloor \alpha \cdot (n-2) \rfloor$, and α is the mutation intensity parameter.

3.2 Ruin and recreate heuristics

h_6: Remove l, $l = \lfloor \alpha \cdot (n-1) \rfloor$, randomly selected jobs and reinsert them in an NEH fashion. This heuristic resembles the main component of the iterated greedy heuristic proposed in [3] for the permutation flow shop and later for the permutation flow shop with sequence dependent setup times [5].

h_7: Remove l, where l is as above, randomly selected jobs, reinsert them in an NEH fashion but this time, at every iteration of the NEH procedure the best q, $q = \lfloor \beta \cdot (l-1) \rfloor + 1$, sequences generated so far are considered for the reinsertion.

3.3 Local search heuristics

h_8: This is a steepest descent local search. At every iteration each job is removed from its current position and assigned into all remaining positions. The job is fixed to the position that leads to the best schedule. This is repeated until no improvement is observed.

Table 1: Instance size

instance	size
0-9	20×5
10-19	20×10
20-29	20×20
30-39	50×5
40-49	50×10
50-59	50×20
60-69	100×5
70-79	100×10
80-89	100×20
90-99	200×10
100-109	200×20
110-119	500×20

h_9: This is a first improvement local search. At every iteration each job is removed from its current position and assigned into the remaining positions. This time, if an improvement movement is found, this is immediately accepted, and the search continues with the next job. This is repeated until no improvement is observed.

h_{10}: This is a random single local search pass. In this, $r = \lfloor \beta(n-1) \rfloor + 1$ randomly selected jobs are tested (one at a time) on all positions and fixed to the best possible place. This is only done once.

h_{11}: This is a first improvement random single local search pass. This is as h_9 but jobs are assigned to the first place that improves the current schedule, i.e. jobs are not necessarily tested in all positions. This is only done once.

3.4 Crossover heuristics

The following crossover heuristics take two permutations as an input and return a new permutation. These operators have been designed for permutation representation problems, including scheduling problems.

h_{12}: Order crossover, [7]

h_{13}: Partially mapped crossover, [8]

h_{14}: Precedence preservative crossover, [9]

h_{15}: One point crossover

4 Problem Instances

The HyFlex permutation flow shop problem domain provides a subset of the hardest instances in the Taillard set [10]. The complete set is given in Table 1, where the second column gives the instance size in the format $n \times m$. The job processing times, on all instances, are uniformly distributed random integers in the range $[1, 99]$.

References

[1] M. Nawaz, E. Enscore Jr., and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA-International Journal of Management Science*, 11(1):91–95, 1983.

[2] R. R. García and C. Maroto. A genetic algorithm for hybrid flow shops with sequence dependent setup times and machine elegibility. *European Journal of Operational Research*, 169:781–800, 2006.

[3] R. Ruiz and T. G. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177:2033–2049, 2007.

[4] J. M. Framinan, R. Leisten, and C. Rajendran. Different initial sequences for the heuristic of nawaz, enscore and ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *International Journal of Production Research*, 41:121–148, 2003.

[5] R. Ruiz and T. Stützle. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143–1159, 2008.

[6] P. J. Kalczynski and J. Kamburowski. On thenehheuristic for minimizing the makespan in permutation flowshops. *OMEGA-International Journal of Management Science*, 35:53–60, 2007.

[7] L. Davis. Job shop scheduling with genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 136–140, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc.

[8] D. E. Goldberg and Jr. R. Lingle. Alleles, loci, and the traveling salesman problem. In *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Hillsdale, New Jersey, 1985. Lawrence Erlbaum Associates.

[9] C. Bierwirth, D. C. Mattfeld, and H. Kopfer. On permutation representations for scheduling problems. In H. Voigt, W. Ebeling, I. Rechenberg, and H. Schwefel, editors, *Proceedings of the Parallel Problem Solving from Nature Conference - PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*. Springer, 1996.

[10] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.