THE TIEBREAKING SPACE OF CONSTRUCTIVE HEURISTICS FOR THE PERMUTATION FLOWSHOP MINIMIZING MAKESPAN

INSTITUTO DE INFORMÁTICA UFRGS

Marcus Ritt, Alexander J. Benavides

GECCO — August 2021

1. Introduction
2. Flow shop scheduling
3. Constructive heuristics for the PFSSP
4. A search method for the space of all tie breakers
5. Computational experiments
6. Conclusions

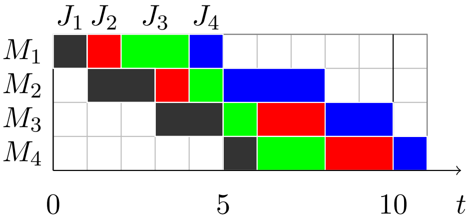
- Permutation flow shop scheduling problem (PFSSP)
  - A basic, NP-hard scheduling problem, many variants with direct practical applications.
- Intensive research on tiebreakers for insertion-based constructive heuristics.
- This motivates two **research questions**:
  - How far are the current best methods from optimal tie breaking?
  - Can we improve current best methods to achieve a performance similar to optimal tie breaking?

Flow shop scheduling

- Flow Shop Scheduling Problem (FSSP)
  - Schedule jobs $J_1, \ldots, J_n$ on machines $M_1, \ldots, M_m$.
  - Job J_i must be processed on machine M_j in time $p_{ij} \geqslant 0$.
  - No preemption.
  - Each machine processes only one job at a time.
  - Objective: minimize the *makespan*.
- Permutation Flow Shop Scheduling Problem (PFSSP)
  - Jobs are processed on all machines in the same order.
- NP-Hard for three or more machines (Garey and Johnson 1979).

Flow shop scheduling – Example

	Machine			
Job	M_1	M_2	M_3	M_4
J_1	1	2	2	1
J_2	1	1	2	2
J_3	2	1	1	2
J_4	1	3	2	1



A schedule is defined by a **permutation** $\pi \in \mathfrak{S}_n$ of $[n]$.

- Then the **completion time** C_{kj} of the kth job π_k of a given job permutation $\pi \in \mathfrak{S}_n$ on machine M_j is given by

$$C_{kj} = \max\{C_{k-1,j}, C_{k,j-1}\} + p_{\pi_k,j} \tag{1}$$

  with boundary conditions $C_{0j} = C_{k0} = 0$, for $j \in [m]$, $k \in [n]$.

- **Makespan** of the schedule: completion time of the last operation $C_{\mathsf{max}} = C_{nm}$.
- Optimization problem: find

$$\pi^* = \operatorname*{argmin}_{\pi \in \mathfrak{S}_n} C_{\mathsf{max}}(\pi)$$

Constructive heuristics for the PFSSP

Constructive heuristics based on job insertion

- Dozens of constructive heuristics for the PFSSP have been proposed in the literature, e.g. Dannenbring (1977), Ho and Chang (1991), Suliman (2000), and Nawaz, Enscore, and Ham (1983).
- The most successful template is based on job insertion:

Input : A job order ρ.
Output : A permutation π.
$\pi = ()$
for $k \in |\rho|$ **do**
(▲) insert ρ_k into π such that $C_{\max}(\pi)$ is minimal
return π

- Example: NEH (Nawaz, Enscore, and Ham 1983).
  - Let $P_i = \sum_{j \in [m]} p_{ij}$ be the total task time of job $i \in [n]$.
  - Process the jobs in order of non-increasing total task times.
  - Namely for job order ρ and $i, j \in [n]$, $i < j$ we have $P_{\rho_i} \geqslant P_{\rho_j}$.

Job orders

Most effective job orders (Fernandez-Viagas, Ruiz, and Framinan 2017):

- SD: non-increasing total task times (Nawaz, Enscore, and Ham 1983);
- AD: mean task times plus standard deviation (Dong, Huang, and Chen 2008);
- ADS: mean task times plus standard deviation plus skewness (Liu, Jin, and Price 2017);
- KK1: by an extension of Johnson's rule for the 2-machine FSSP (Kalczynski and Kamburowski 2008);
- KK2: (Kalczynski and Kamburowski 2009)

Tie breaker

Ties occur frequently when inserting jobs. This motivates tie breakers. Most effective tie breakers:

- FS: first slot;
- LS: last slot;
- TBKK2: (Kalczynski and Kamburowski 2008);
- TBKK3: (Kalczynski and Kamburowski 2009);
- FF: approximation of the idle time without back delays (Fernandez-Viagas and Framinan 2014)

A search method for the space of all tie breakers

Searching in the tie breaking space

- We enumerate all tied solutions.
- To find good solutions early we propose a cyclic best-first search (CBFS) (Kao, Sewell, and Jacobson 2009).
- CBFS visits cyclically solutions of all depths, and in each depth always selects the current best solution.

Cyclic best-first search

Input : A job order π.
initialize priority queues $q_0 = \{()\}, q_1 = \cdots = q_{n-1} = \varnothing$
maintain the best solution π^* during the search
$d := 0$
while $q_d \neq \varnothing$ **do**
    $\rho :=$ deletemin(q_d)
    $S := \{\rho \downarrow^i \pi_{d+1} \mid 0 \leqslant i \leqslant d\}$
    $C^* := \min_{\pi \in S}\{C_{\max}(\pi)\}$
(▪)   **if** $C^* > C_{max}(\pi^*)$ **then**
        continue
    $S^* := \{\pi \in S \mid C_{\max}(\pi) = C^*\}$
    if *d+1=n* **then**
        evaluate all solutions in S^*
(▴)   optionally stop after visiting $S_{\max}$ solutions
    else
(○)   optionally empty q_{d+1} if C^* is the new best makespan on level $d+1$
        insert all solutions in S^* into q_{d+1}
(•)   optionally limit the queue size to $Q_{\max}$ solutions
    advance d cyclically to the next value such that $q_d \neq \varnothing$, if any

Cyclic best-first search

Input : A job order π.
initialize priority queues $q_0 = \{()\}, q_1 = \cdots = q_{n-1} = \emptyset$
maintain the best solution π^* during the search
$d := 0$
while $q_d \neq \emptyset$ **do**
    $\rho :=$ deletemin(q_d)
    $S := \{\rho \downarrow^i \pi_{d+1} \mid 0 \leqslant i \leqslant d\}$
    Pruning: discard worse solutions
    $C^* := \min_{\pi \in S}\{C_{\max}(\pi)\}$
(■)   **if** $C^* > C_{max}(\pi^*)$ **then**
        continue
    $S^* := \{\pi \in S \mid C_{\max}(\pi) = C^*\}$
    if $d+1=n$ **then**
        evaluate all solutions in S^*
(▲)   optionally stop after visiting $S_{\max}$ solutions
    else
(○)   optionally empty q_{d+1} if C^* is the new best makespan on level $d+1$
        insert all solutions in S^* into q_{d+1}
(•)   optionally limit the queue size to $Q_{\max}$ solutions
    advance d cyclically to the next value such that $q_d \neq \emptyset$, if any

Cyclic best-first search

Input : A job order π.
initialize priority queues $q_0 = \{()\}, q_1 = \cdots = q_{n-1} = \emptyset$
maintain the best solution π^* during the search
$d := 0$
while $q_d \neq \emptyset$ **do**
    $\rho :=$ deletemin(q_d)
    $S := \{\rho \downarrow^i \pi_{d+1} \mid 0 \leqslant i \leqslant d\}$
Pruning: discard worse solutions
    $C^* := \min_{\pi \in S}\{C_{\max}(\pi)\}$
(▪)   **if** $C^* > C_{max}(\pi^*)$ **then**
        continue
    $S^* := \{\pi \in S \mid C_{\max}(\pi) = C^*\}$
    if $d+1=n$ **then**
        evaluate all solutions in S^*
(▲)   optionally stop after visiting $S_{\max}$ solutions
Aggressive pruning: discard non locally-optimal solutions
    else
(○)   optionally empty q_{d+1} if C^* is the new best makespan on level $d+1$
        insert all solutions in S^* into q_{d+1}
(•)   optionally limit the queue size to $Q_{\max}$ solutions
    advance d cyclically to the next value such that $q_d \neq \emptyset$, if any

Cyclic best-first search

Input : A job order π.
initialize priority queues $q_0 = \{()\}, q_1 = \cdots = q_{n-1} = \emptyset$
maintain the best solution π^* during the search
$d := 0$
while $q_d \neq \emptyset$ **do**
  $\rho :=$ deletemin(q_d)
  $S := \{\rho \downarrow^i \pi_{d+1} \mid 0 \leqslant i \leqslant d\}$
Pruning: discard worse solutions
  $C^* := \min_{\pi \in S}\{C_{\max}(\pi)\}$
(■)   **if** $C^* > C_{max}(\pi^*)$ **then**
    continue
  $S^* := \{\pi \in S \mid C_{\max}(\pi) = C^*\}$
  if $d+1=n$ **then**
Limit number of visited global solutions
    evaluate all solutions in S^*
(▲)   optionally stop after visiting $S_{\max}$ solutions
  else
Aggressive pruning: discard non locally-optimal solutions
(○)   optionally empty q_{d+1} if C^* is the new best makespan on level $d+1$
    insert all solutions in S^* into q_{d+1}
(•)   optionally limit the queue size to $Q_{\max}$ solutions
  advance d cyclically to the next value such that $q_d \neq \emptyset$, if any

Cyclic best-first search

Input : A job order π.
initialize priority queues $q_0 = \{()\}, q_1 = \cdots = q_{n-1} = \emptyset$
maintain the best solution π^* during the search
$d := 0$
while $q_d \neq \emptyset$ **do**
    $\rho :=$ deletemin(q_d)
    $S := \{\rho \downarrow^i \pi_{d+1} \mid 0 \leqslant i \leqslant d\}$
Pruning: discard worse solutions
    $C^* := \min_{\pi \in S}\{C_{\max}(\pi)\}$
(▪)   **if** $C^* > C_{max}(\pi^*)$ **then**
        continue
    $S^* := \{\pi \in S \mid C_{\max}(\pi) = C^*\}$
    if $d+1=n$ **then**
Limit number of visited global solutions
        evaluate all solutions in S^*
(▲)   optionally stop after visiting $S_{\max}$ solutions
    else
Aggressive pruning: discard non locally-optimal solutions
(○)   optionally empty q_{d+1} if C^* is the new best makespan on level $d+1$
Limit number of visited local solutions
        insert all solutions in S^* into q_{d+1}
(•)   optionally limit the queue size to $Q_{\max}$ solutions
    advance d cyclically to the next value such that $q_d \neq \emptyset$, if any

Computational experiments

Test instances and methodology

- Experiments:
  1. What is the potential of tie breakers?
  2. How does solution quality improve over time?
  3. Can the cyclic best-first replace existing heuristics?
- A total of 669 instances.

Source	N	n	m
Carlier (1978)	8	7,8,10,11,12,13,14	4,5,6,7,8,9
Demirkol (1998)	40	20,30,40,50	15,20
Reeves (1995)	21	20,30,50,75	5,10,15,20
Taillard (1993)	120	20,50,100,200,500	5,10,20
Vallada et al. (2015)	480	$10, 20, \ldots, 60, 100, 200, \ldots, 800$	5,10,15,20,40,60

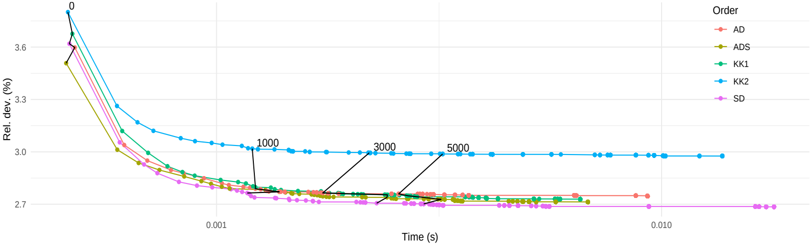
Experiment 1: the potential of perfect tie breaking

- Enumerate all solutions for job orders NEH, AD, ADS, KK1, and KK2.
- 184 instances could be solved for all job orders.
- Tie breakers have strong effect on quality: relative deviations from the best known values range from abt. $2.7\,\%$ to $5.4\,\%$.
- Plenty of room for improvement: ideal tie breaker can produce results that are abt. $1\,\%$ better than current best.

			Rel. dev. (%)			Tie breakers				
Order	$S\ (K)$	$t\ (s)$	min	mean	max	FF	TBKK2	TBKK3	FS	LS
SD	627.2	7.71	2.68	4.03	5.33	3.62	3.70	3.74	3.87	3.83
ADS	631.5	1.90	2.71	3.95	5.13	3.51	3.66	3.64	3.62	3.77
KK1	549.3	4.02	2.73	4.03	5.26	3.68	3.70	3.69	3.93	3.87
AD	557.2	4.82	2.75	3.94	5.02	3.60	3.73	3.69	3.68	3.70
KK2	435.8	13.09	2.97	4.20	5.37	3.80	3.85	3.87	3.92	4.02

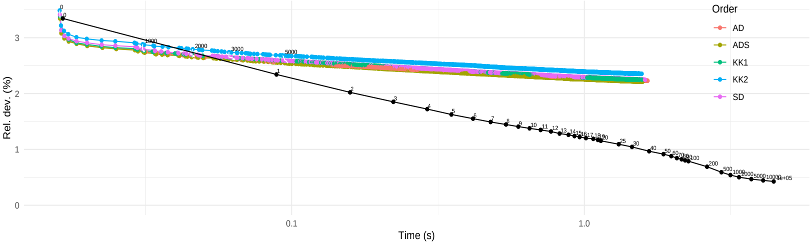
Experiment 2: temporal evolution of makespans

- Limit of 100K partial solutions.
- Relative deviations over time for instances that can be enumerated completely for five different job orders (without/with pruning: $184/205$).
- Sharp drop in the beginning and then slowly convergence to best possible.
- After 0.003 seconds or abt. 5K nodes, solutions are essentially equal to the best possible.



Experiment 3: cyclic best-first search as a heuristic

- Comparison to the temporal evolution of the state-of-the-art iterated greedy heuristic.
- Exploring the space of all tie breakers is advantageous for small time scales up to about 2000 processed solutions or an average time of about 0.05s.



Conclusions

- Effective tie breaking has a large potential for improving solutions, even for fixed job orders.
- Average relative deviation of the current best tie breaker of about $3.5\,\%$ can be improved to about 2%.
- Most of this potential can be exploited in a short time, using a cyclic best-first search.

Thanks for your attention!

References I

- Dannenbring, David G. (1977). "An Evaluation of Flow Shop Sequencing Heuristics". In: *Management Science* 23.11, pp. 1174–1182. DOI: 10.1287/mnsc.23.11.1174.
- Dong, X., H. Huang, and P. Chen (2008). "An improved NEH-based heuristic for the permutation flowshop problem". In: *Computers & Operations Research* 35, pp. 3962–3968. DOI: 10.1016/j.cor.2007.05.005.
- Fernandez-Viagas, Victor and Jose M. Framinan (2014). "On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem". In: *Computers & Operations Research* 45, pp. 60–67. DOI: 10.1016/j.cor.2013.12.012.

References II

- Fernandez-Viagas, Victor, Rubén Ruiz, and Jose M. Framinan (2017). "A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation". In: *European Journal of Operational Research* 257.3, pp. 707–721. DOI: 10.1016/j.ejor.2016.09.055.
- Garey, Michael R. and David S. Johnson (1979). *Computers and intractability: A guide to the theory of NP-completeness*. Freeman.
- Ho, Johnny C. and Yih-Long Chang (1991). "A new heuristic for the n-job, m-machine flow-shop problem". In: *European Journal of Operational Research* 52.2, pp. 194–202. DOI: 10.1016/0377-2217(91)90080-F.

References III

- Kalczynski, Pawel Jan and Jerzy Kamburowski (2008). "An improved NEH heuristic to minimize makespan in permutation flow shops". In: *Computers & Operations Research* 35, pp. 3001–3008. DOI: 10.1016/j.cor.2007.01.020.
- – (2009). "An empirical analysis of the optimality rate of flow shop heuristics". In: *European Journal of Operational Research* 198, pp. 93–101. DOI: 10.1016/j.ejor.2008.08.021.
- Kao, Gio K., Edward C. Sewell, and Sheldon H. Jacobson (2009). "A branch, bound, and remember algorithm for the $1 \mid r_i \mid \sum t_i$ scheduling problem". In: *Journal of Scheduling* 12.163. DOI: 10.1007/s10951-008-0087-3.

References IV

- Liu, Weibo, Yan Jin, and Mark Price (2017). "A new improved NEH heuristic for permutation flowshop scheduling". In: *International Journal of Production Economics* 193, pp. 21–30. DOI: 10.1016/j.ijpe.2017.06.026.
- Nawaz, M., E.E. Enscore, and I. Ham (1983). "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem". In: *Omega* 11.1, pp. 91–95. DOI: 10.1016/0305-0483(83)90088-9.
- Suliman, S. M. A. (2000). "A two-phase heuristic approach to the permutation flow-shop scheduling problem". In: *International Journal of Production Economics* 64.1–3, pp. 143–152. DOI: 10.1016/S0925-5273(99)00053-5.