A SIMPLE, ADAPTIVE BUBBLE SEARCH FOR IMPROVING HEURISTIC SOLUTIONS OF THE PERMUTATION FLOW SHOP SCHEDULING PROBLEM

Tadeu Zubaran and Marcus Ritt
Departamento de Informática Teorica, Instituto de Informática
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
{tkzubaran, marcus.ritt}@inf.ufrgs.br

ABSTRACT

In this paper we study the application of Bubble Search, an extension of priority-based greedy heuristics proposed by Lesh and Mitzenmacher (2006), as an improvement procedure for heuristic solutions of the permutation flow shop scheduling problem. We compare the performance of Bubble Search on different time scales and for different parameter settings. In particular, we demonstrate the optimal parameter setting depends on the size of the instance, and propose a simple adaptive extension of Bubble Search. Computational experiments demonstrate that the adaptive version outperforms fixed parameter settings. They further show that adaptive Bubble Search is competitive with the most effective constructive heuristics in a comparable time scale, and therefore is a promising technique for flow shop scheduling and related problems.

KEYWORDS. Flow shop scheduling. Heuristics. Bubble Search.

Combinatorial optimization

RESUMO

Neste artigo nós estudamos a aplicação do Bubble Search, uma extensão de algoritmos baseados em prioridade proposto por Lesh and Mitzenmacher (2006), como um procedimento de melhora para soluções heurísticas do problema permutation flow shop. Nós comparamos a performance do Bubble Search em diferentes escalas de tempo e para diferentes conjuntos de valores para os parâmetros. Em particular, nós demonstramos que os valores ótimos para parâmetros dependem do tamanho da instância, e propomos uma simples extensão adaptativa para o Bubble Search. Experimentos computacionais demonstram que a versão adaptativa tem performance superior para um conjunto fixo de parâmetros. Elas também mostram que o Bubble Search adaptativo é competitivo com as heurísticas construtivas mais efetivas em escala de tempo comparável, e portanto é uma técnica promissora para o problema de agendamento flow shop e problemas relacionados.

PALAVRAS CHAVE. Escalonamento de tarefas, Heurísticas, Bubble search.

Otimização combinatória

1. Introduction

In the *flow shop scheduling problem* (FSSP) we have to find an optimal schedule to process a set of jobs $J = [n]$ on a set of machines $M = [m]$[1]. Job $j \in J$ has a processing time of p_{ij} on machine $i \in M$. Each machine can process only one job at time, and each job can be processed only on one machine at any instant. The jobs have to be processed without preemption. In a flow shop each job has to be processed on all the machines in the fixed order $1, \ldots, m$. We consider no explicit setup times, yet since there is no preemption we can assume that the setup is included in the processing time. There are several possible objective functions, e.g. total flow time, or total lateness or tardiness. Here we focus on the most common objective, which is to minimize the *makespan* $C_{\max}$, i.e. the time of completion of the last job.

The *permutation flow shop scheduling problem* (PFSSP) is a widely studied variation of the FSSP. This model restricts the schedules to *permutation schedules*. In a permutation schedule, all jobs have to be processed on all machines in the same order, which reduces the number of possible solutions to $n!$. According to the classification proposed by Graham et al. (1979) the PFSSP with the objective to minimize the makespan is denoted by $F \mid \mathrm{prmu} \mid C_{\max}$.

For more than two machines the PFSSP is a NP-hard problem (Kan, 1976) and since the seminal work by Johnson (1954) who proposed a polynomial algorithm for the case $m = 2$ it has been studied thoroughly. Instances of size useful in actual applications are usually not exactly solvable in reasonable time, therefore several constructive and improvement heuristics have been proposed. In particular we have the simple and surprisingly effective constructive heuristic NEH (Nawaz et al., 1983) which is often used to generate initial solutions for more sophisticated approaches. Taillard (1990) has shown that the NEH heuristic can be implemented in time $O(n^2m)$. It is well known that tie-breaking rules are important for the performance of NEH-like heuristics. Kalczynski and Kamburowski (2008) study such rules and propose the improved constructive heuristic NEHKK1. Extensions of the NEH-like heuristics, which try to overcome the fixed job order, have been studied by Farahmand et al. (2009).

Framinan et al. (2003) considers adaptations of the NEH heuristic when the objective is other than minimize the makespan. Other works with notable results are the ant colony algorithm of Rajendran and Ziegler (2004), the CDS algorithm (Campbell et al., 1970), the work from Dannenbring (1977) which performs a constructive heuristic followed by an improvement phase, the iterated greedy algorithm from Ruiz and Stützle (2007), and the tabu search of Nowicki and Smutnicki (1996). For more details on metaheuristic approaches to solve the PFSSP, we refer the reader to the excellent surveys of Gupta and Stafford (2006) and Potts and Strusevich (2009).

In this work we study the application of Bubble Search as an improvement procedure for heuristic solutions based on NEH for the PFSSP with the objective of minimizing the makespan. Our study is motivated by the fact that Bubble Search probabilistically examines small modifications of a given base order, and is therefore well suited for improving priority-based greedy algorithms.

The remainder of this paper is organized as follows. In Section 2 we explain the NEH heuristic of Nawaz et al. (1983) and its best performing variants. Section 3 explains deterministic and randomized Bubble Search and variants thereof. In Section 4 we discuss the dependency of Bubble Search on the size of the instance and propose a new variant called *adaptive Bubble Search* which takes this dependency into account. We present the results of computational experiments in Section 5, and conclude in Section 6.

[1] We use the notation $[n] = \{1, \ldots, n\}$.

Algorithm 1 NEH

Input: Processing times $p_{ij}\ \forall i \in M, j \in N$.
let $P_j := p_{1j} + p_{2j} + \cdots + p_{mj}$.
order the jobs such that $P_1 \geq P_2 \geq \cdots \geq P_n$.
let $\sigma = ()$ be the empty sequence.
for all jobs $j = 1, \ldots, n$ **do**
    insert j in σ in the sequence at the position where it yields the lowest makespan.
return σ

2. Constructive heuristics for the PFSSP

NEH and its variants are greedy constructive algorithms. For a given job order, they start with an empty schedule and insert the next job in that order into the current partial schedule at the position which maintains the makespan of the current partial schedule shortest. Thus, such an algorithm performs n insertions. When inserting the jth job, the algorithm must determine the makespan of $j-1$ candidate insertion points, which leads to an overall of $\binom{n}{2} = \Theta(n^2)$ makespan computations. Computing a makespan in $O(nm)$ yields a $O(n^3m)$ algorithm. Taillard (1990) has shown that the makespan of all insertion points of a fixed job can be computed in time $O(nm)$ which reduces the time complexity to $O(n^2m)$. The basic NEH algorithm as proposed by Nawaz et al. (1983) processes the jobs in order of non-increasing total processing times $\sum_{i\in[m]} p_{ij}$, for $j \in [n]$ and is shown in Algorithm 1.

Kalczynski and Kamburowski (2008) observed that the NEH heuristic can be improved by inserting the jobs in the same order as proposed by Johnson (1954) for the two-machine permutation flow shop. If we define

$$\hat{a_j} = \sum_{i\in[m]} \left(\binom{m-1}{2} + m - i \right) p_{i,j}$$
$$\hat{b_j} = \sum_{i\in[m]} \left(\binom{m-1}{2} + i - 1 \right) p_{i,j}$$

then NEHKK1 insert the jobs in order of non-increasing $\hat{c}_j = \min\{\hat{a}_j, \hat{b}_j\}$. Ties between different insertion positions are broken by choosing the first job of minimum makespan, if $\hat{c}_j = \hat{a}_j$, and the last job otherwise. NEHKK1 reduces the average percent relative deviation from the best known values in the instances proposed by Taillard (1993) by about 15%. The NEHKK1 heuristic has the additional property that it solves two-machine permutation flow shops optimally. We use the result from the NEHKK1 heuristic as the starting point for the improvement with Bubble Search, as described below.

Farahmand et al. (2009) proposed several extensions to NEH-like heuristics. The variants that perform best reinsert the previously inserted jobs at positions $\max\{1, p-k\}, \ldots, \min\{p+k, j\}$ again after the insertion of the jth job at position p, for a parameter k. Their algorithm FRB3 which considers all previously inserted jobs is equivalent to FRB4_n. FRB4_k is shown in Algorithm 2. The reinsertion of the jobs is designed to counteract the strong greediness of NEH. FRB4_k has worst case time complexity $O(kn^2m)$.

The last algorithm we consider is called *FRB5*, also proposed by Farahmand et al. (2009), which consists of the simple idea of performing the algorithm NEH followed by a full local search after the insertion of each new job in the insertion neighborhood N_1. This neighborhood considers all possible reinsertion points of all jobs, similar to FRB3. The local search stops when a local minimum is reached.

Algorithm 2 FRB4_k

Input: Processing times $p_{ij}\ \forall i \in M, j \in N$.
let $P_j := p_{1j} + p_{2j} + \cdots + p_{mj}$.
order the jobs such that $P_1 \geq P_2 \geq \cdots \geq P_n$.
let $\sigma = ()$ be the empty sequence.
for all jobs $j = 1, \ldots, n$ **do**
  insert j in σ in the sequence at the position where it yields the lowest makespan.
  for all positions $i = \max\{1, p-k\}, \ldots, \min\{p+k, j\}$ **do**
    reinsert the job at position i at the position which yields the lowest makespan.
return σ

3. Bubble Search and its variants

Bubble Search and its variants have been proposed by Lesh and Mitzenmacher (2006) as simple and flexible modifications procedures for priority-based construction algorithms. For a solution space which consists of all $n!$ permutations like in the PFSSP, visiting all of them exhaustively will eventually find the optimal solution. The exhaustive search can be performed in any order. In particular, *exhaustive Bubble search* proposes to visit all the permutations in order of increasing Kendall-tau distance from a given permutation. Ties may be broken arbitrarily. Formally, for two n-permutations π and π' the Kendall-tau distance, also known as the Bubble Sort distance, is defined as

$$d(\pi, \pi') = \sum_{1 \leq i < j \leq n} [\pi(i) < \pi(j) \text{ and } \pi'(i) > \pi'(j)].$$

It measures the number of transpositions necessary to transform π into π'. Therefore $d(\pi, \pi') \in [0, \binom{n}{2}]$.

The search can be stopped at any time and will return best permutation found so far. The performance of this truncated Bubble Search depends on a good initial order and a problem structure, that makes more likely that good solutions are close to this initial order, such that it is more likely that we have visited solutions with better quality than random ones.

A natural variation of this proposal is the *randomized Bubble Search*, in which we choose random permutations with a probability that decreases with increasing Kendall-tau distance. Lesh and Mitzenmacher (2006) suggest that we use a probability proportional to $(1-\alpha)^{-d}$ where α is a parameter to be adjusted and d the Kendall-Tau distance to the base ordering. For $\alpha = 0$ randomized Bubble Search degenerates into random sampling, and for $\alpha = 1 - \epsilon$ it approximates a sampling in a neighborhood that allows only swaps of two adjacent elements in the base order.

A further improvement of the randomized Bubble Search is the *randomized Bubble Search with replacement*. This strategy replaces the current ordering by the new ordering, whenever the new ordering is better than the current one. This turns randomized Bubble Search into a stochastic search algorithm. Lesh and Mitzenmacher (2006) observed to Bubble search with replacement typically outperforms the simpler variants of Bubble Search. They also have demonstrated that Bubble Search can produce results that are competitive with similar GRASP-based algorithms.

4. An adaptive variant of Bubble Search

Randomized Bubble Search can be implemented by a very simple stochastic process. To select a new ordering, we start with an empty ordering, and process the elements of the current base ordering. With probability α we select the current element, remove it from the base ordering and append it to the new ordering. In this case we start visiting the elements from the start. Otherwise,

Algorithm 3 Bubble Search sampling

Input: A base permutation σ^*, a probability α.
let $\sigma = ()$ be the empty sequence.
for $i = 1, \ldots, n$ **do**
  $j := 1$
  while no element selected **do**
    with probability α: select element σ^*_j
    otherwise: $j := j \mod n + 1$
  remove σ^*_j from σ^* and append it to σ
return σ

with probability $1 - \alpha$, we continue with the next element of the current base ordering, cycling as necessary. This process is repeated until the base ordering is empty and is shown in Algorithm 3.

This process guarantees that an ordering of Kendall-tau distance d is selected with probability proportional to $(1-\alpha)^d$ (Lesh and Mitzenmacher, 2006). However, since the range of the Kendall-tau distance increases with n, the probability of selecting orderings with a fixed distance from the current ordering decreases with increasing n. We will show below that this behaviour can affect the scalability of Bubble Search adversely. We therefore propose to adjust α as a function of n as follows.

Consider the probability $P[d = 0]$ that Algorithm 3 returns σ^* on input σ^*. We have

$$\begin{aligned} P[d=0] &= \alpha(1 + (1-\alpha)^n + (1-\alpha)^{2n} + \ldots) \\ &\quad \times \alpha(1 + (1-\alpha)^{n-1} + (1-\alpha)^{2(n-1)} + \ldots) \\ &\quad \times \cdots \times \alpha(1 + (1-\alpha)^2 + (1-\alpha)^4 + \ldots) \times 1 \\ &= \prod_{i \in [n]} \frac{\alpha}{1-(1-\alpha)^i} = O(\alpha^n). \end{aligned}$$

Therefore, we propose to compensate this behaviour by setting $\alpha = \alpha_0^{1/n}$ for some base probability α_0 which has to be calibrated. We call this approach *adaptive randomized Bubble Search*. Adaptive Bubble Search can be applied with or without replacement.

5. Computational results

In this section we report on computational experiments done with several variants of Bubble Search starting from the base ordering obtained by the NEHKK1 algorithm presented in Section 2. All experiments have been done on a PC with an Intel Core i7 930 processor running at 2.80 GHz and 12 GB of main memory. In the experiments only one core of the processor has been used. We have implemented the NEHKK1 algorithm using the enhancements proposed by Taillard (1990) to run in $O(n^2m)$ and the standard and adaptive variant of randomized Bubble Search in C++. The code has been compiled with GNU C++ compiler version 4.6.3 and maximal optimization.

For our test use the well known and widely used set of instances proposed by Taillard (1993). The test set consists of 12 groups of instances with the number of jobs varying between 20 and 500, and the number of machines between 5 and 20. Each group contains 10 instances, with processing times drawn uniformly at random in the interval $[1, 99]$. The current best known values for all 120 instances can be obtained at Taillard (2013).

5.1. An analysis of the N_1 neighborhood on the instance Carlier5

To decide the best acceptance criterion for Bubble Search with replacement we first conducted an analysis of the N_1 neighborhood. The analysis has been done in the instance 5 proposed by Carlier

Table 1. Characterization of all solutions of instance Carlier5.

Type of solution	Number	%
Isolated	0	0
Strict local maximum	0	0
Plateau	0	0
Local maximum	6	0.00017
Strict local minimum	5	0.00014
Slope	134784	3.71
Local minimum	1743	0.048
Ledge	3492262	96.24

(1978), since it has only 10 jobs and thus can be analyzed exhaustively. A classification of the $10! = 3.628.800$ solutions can be seen in Table 1.

We can observe that the majority of the solutions are ledges (i.e. they have better, equal, and worse neighbors), followed by slopes. All three optimal solutions of value 7720 are non-strict local minima. Based on this analysis, we opted to not only accept solutions that are strictly better in Bubble Search with replacement, but also solutions of equal value. Preliminary tests have shown that this strategy performs significantly better than accepting only strictly better solutions.

5.2. Parameter adjustment

We next investigated the dependency of randomized Bubble Search with replacement on the parameter α. We selected the first instance in each group, and ran Bubble Search with a time limit of $nm/2 \times 60\text{ms}$ for $\alpha \in \{0.6, 0.7, 0.8, 0.9, 0.95\}$. Each test was replicated three times. The results of these tests can been seen in Table 2. It reports for each value of α the average percent relative deviation from the best known value over all 12 test instances. The smallest deviations are highlighted in boldface.

The overall best value is $\alpha = 0.9$ with a relative deviation of 1.41. There is however a clear tendency to perform better with larger values of α for an increasing number of jobs. We therefore chose a base value of $\alpha_0 = 8 \times 10^{-4}$ and ran adaptive Bubble Search with $\alpha = \alpha_0^{1/n}$. The choice of α_0 fixes $\alpha = 0.7$ for $n = 20$. The results for adaptive Bubble Search can be seen in the last column (Adapt). Adaptive Bubble Search performs clearly better than any Bubble Search with a fixed alpha, obtaining an overall average percent relative deviation of 1.24. The relative deviation for each instance size is close to the best for the individual α values.

Table 2 also contains in column "NEHKK1" the relative deviations of the initial solutions obtained by NEHKK1 from the best known value. We can see that randomized Bubble Search with replacement has the potential to substantially improve over the values found by NEHKK1.

5.3. Experiments on the full test set

Based on the parameter adjustment above, we ran adaptive Bubble Search on all 120 instances. We tested the method on three time scales, namely $nm/2 \times 0.5\text{ms}$, $nm/2 \times 8\text{ms}$, and $nm/2 \times 60\text{ms}$ to evaluate independently its utility for quickly improving initial solutions, as well as its limit in the long term. Each test was replicated five times.

The results of these experiments can be seen in Table 3. It reports the average percent relative deviation from the best known values for each group of instances for the original NEH heuristics, the improved NEHKK1, the algorithms FRB3, FRB4_{12}, and FRB5 proposed by Farahmand et al. (2009), and for Bubble Search for the three time scales above.

Table 2. Results for randomized Bubble Search with replacement for different values of α.

			α (%)					
n	m	NEHKK1	60	70	80	90	95	Adapt.
20	5	1.41	0.00	0.00	0.00	0.00	0.00	0.00
20	10	6.19	**0.46**	0.51	0.91	1.10	1.41	0.52
20	20	4.66	1.34	**1.00**	1.23	1.61	2.25	1.04
50	5	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	10	5.95	5.02	4.51	2.75	**1.49**	2.84	2.66
50	20	5.22	4.48	3.97	3.24	**3.03**	3.10	3.14
100	5	0.38	0.00	0.00	0.00	0.00	0.00	0.00
100	10	1.32	0.88	0.63	0.18	0.20	**0.16**	0.16
100	20	5.47	5.47	5.43	5.07	3.85	**3.30**	3.41
200	10	0.97	0.68	0.33	0.13	**0.09**	0.21	0.16
200	20	3.22	3.22	3.22	3.22	2.92	**2.41**	2.24
500	20	2.71	2.71	2.71	2.71	2.57	**2.14**	1.54
Averages		3.12	2.02	1.86	1.62	**1.41**	1.49	1.24

Values are averages over three replicates.

Table 3. Results for adaptive randomized Bubble Search with replacement for time scales $nm/2 \times t_0$, $t_0 \in \{0.5, 8, 60\}$ on the complete Taillard instance set.

							Bubble Search			BM
n	m	NEH	NEHKK1	FRB3	$FRB4_{12}$	FRB5	0.5	8	60	8
20	5	3.35	2.81	0.89	1.12	1.08	0.83	0.28	0.22	0.29
20	10	5.02	4.43	1.86	1.79	2.19	2.49	1.77	1.02	1.53
20	20	3.73	3.34	2.18	2.08	1.80	2.24	1.58	1.21	1.54
50	5	0.84	0.67	0.22	0.30	0.19	0.21	0.13	0.09	0.13
50	10	5.12	5.46	2.99	2.89	2.25	3.90	2.76	2.00	2.77
50	20	6.32	6.25	3.38	4.05	3.38	5.08	4.58	4.22	4.58
100	5	0.46	0.43	0.21	0.28	0.16	0.20	0.12	0.08	0.12
100	10	2.13	1.78	0.94	1.22	0.80	1.24	0.84	0.56	0.85
100	20	5.23	5.23	2.90	3.73	2.51	4.58	4.07	3.47	4.07
200	10	1.43	1.11	0.52	0.60	0.38	0.79	0.47	0.40	0.48
200	20	4.52	4.10	2.41	2.95	1.84	3.44	2.95	2.40	2.96
500	20	2.24	2.04	1.06	1.40	0.72	1.67	1.34	1.12	1.34
Averages		3.37	3.14	1.63	1.87	1.44	2.22	1.74	1.40	1.72

Values are averages over three replicates and ten instances.

The experiments confirm the potential of Bubble Search to produce good solutions and can considerably improve over the initial solution obtained by NEHKK1. As expected, the average quality of the solutions improves over time from 2.22% to 1.40%. In average over all instances, in the largest time scale, Bubble Search is able to outperform the best re-insertion heuristic FRB5, although only by a small margin. In the middle time scale its values are competitive in quality with the construction heuristics FRB3 and FRB4_{12}. For very short execution times, it still improves the initial values of NEHKK1 by about 1%, but produces about 0.4% to 0.6% higher deviations than other constructive heuristics.

These results have to be judged relative to the computational effort of the different methods. The result of Farahmand et al. (2009) were obtained on a PC with a Pentium IV processor running at 3.2 GHz, which is comparable to but slower than our machine. The time of Bubble Search, on the other hand, is dominated by the computation of the new makespan, and is not fully optimized, since the application of Taillard's improvements are not straightforward to apply to it. The average execution time of FRB3, FRB4_{12} and FRB5 as reported by Farahmand et al. (2009) is 2.43, 0.20, and 4.88 seconds, respectively, while Bubble Search needs 0.31, 4.51, and 31.60 seconds on the three time scales. Thus, Bubble Search on the small time scale is comparable to FRB4_{12}, on the middle time scale to FRB3 and FRB5.

Looking at the individual instance groups, we can observe that Bubble Search outperforms FRB3 in eight of the 12 groups, in particular for the instances of smaller size. The same holds for three of the smaller instance groups when comparing the short time scale with FRB4_{12}. This indicates that Bubble Search may have a potential advantage on short time scales and small instances. For large instances Bubble Search takes more time to obtain good results. This can be explained by the stochastic component of the method, and the need to compute the makespan of the perturbed orderings from scratch.

We finally briefly evaluated if a milder acceptance criterion would be able to improve the results for Bubble Search in the middle time scale. To this end, we used a Metropolis acceptance criterion, that for temperature T accepts solutions that are worse by Δ with probability $e^{-\Delta/T}$. We followed Ruiz and Stützle (2007) and chose a fixed base temperature of $\overline{p}/10 \times 0.5$, where $\overline{p}$ is the average processing time over all machines and jobs. We adjusted this temperature such that it is about ten times lower for the largest instances. We ran Bubble Search for $750nm$ iterations, and then enabled the Metropolis acceptance criterion. The result of this test can be seen in the final column (BM) of Table 3. The value when using the extended acceptance criterion is better than Bubble Search in the middle time scale, but only slightly. In two of the smaller instances, it obtains better results earlier. In summary, Bubble Search seems relatively robust with respect to the acceptance criterion.

6. Conclusion

In this paper we have investigated the utility of Bubble Search for improving solution for the flow shop scheduling problem. We have analyzed the dependence of Bubble Search on its parameter α and proposed a new adaptive variant of Bubble Search, that scales the probability of accepting a solution of Kendall-tau distance d in accordance with the size of the instance.

In computational experiments we could demonstrate that for flow shop scheduling the adaptive variant outperforms the regular randomized Bubble Search with replacement. The experiments show also that Bubble Search is a promising technique for obtaining results comparable to the best constructive heuristics, especially on small instances.

To realize the full potential of Bubble Search further research is necessary. Based on the computational results, it seems most promising for improving already well optimized instances. For larger instances it should be combined with more direct techniques, for obtaining good starting solution

in a short time. Applying Bubble Search to reduced neighborhoods (e.g. N_5) and for perturbing solutions may be promising research avenues.

References

Campbell, H. G., Dudek, R. A., and Smith, M. L. (1970), A heuristic for the n job, m machine sequencing problem., *Management Science 16(10)*.

Carlier, J. (1978), Ordonnancements a contraintes disjonctives, *R.A.I.R.O. Recherche operationelle/Operations Research*, 12(4):333–351.

Dannenbring, D. G. (1977), An evaluation of flow shop sequencing heuristics., *Management Science July 1977 vol. 23*.

Farahmand, S., Ruiz, R., and Boroojerdian, N. (2009), New high performing heristics for minimizing makespan in permutation flowshops, *Omega, The International Journal of Management Science*.

Framinan, J. M., Leister, R., and Rajendran, C. (2003), 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(1):121-148*.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. H. G. R. (1979), Optimization and approximation in deterministic sequencing and scheduling, *Annals of Discrete Mathematics*, 5:287–326.

Gupta, J. and Stafford, E. (2006), A comprehensive review and evaluation of permutation flowshop heuristics, *Eur. J. Oper. Res.*, 169(3):699–711.

Johnson, S. M. (1954), Optimal two- and three-stage production schedules with setup times included., *Naval Research Logistics Quarterly*.

Kalczynski, P. J. and Kamburowski, J. (2008), An improved neh heuristic to minimize makespan in permutation flow shops, *Computer & Operations Research 35 3001-3008*.

Kan, R. (1976), *Machine Scheduling Problems: Classification, Complexity and Computations*, Martinus Nijhoff, The Hague, The Netherlands.

Lesh, N. and Mitzenmacher, M. (2006), Bubblesearch: A simple heuristic for improving priority-based greedy algorithms, *Information Processing Letters*, 97(4):161 – 169.

Nawaz, M., Enscore Jr, E. E., and Ham, I. (1983), A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, *Omega*, 11(1):91–95.

Nowicki, E. and Smutnicki, C. (1996), A fast tabu search algorithm for the permutation flow-shop problem, *Eur. J. Oper. Res.*, 91(1):160–175.

Potts, C. N. and Strusevich, V. A. (2009), Fifty years of scheduling: a survey of milestones, *J. Oper. Res. Soc.*, 60:S41–S68.

Rajendran, C. and Ziegler, H. (2004), Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs, *European Journal of Operational Research*, 155(2):426 – 438, Financial Risk in Open Economies.

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

Taillard, E. (1990), Some efficient heuristic methods for the flow shop scheduling problem, *European Journal of Operations Research 47 3001-3008*.

Taillard, E. (1993), Benchmarks for basic scheduling problems, *European Journal of Operations Research 64 278-85*.

Taillard, E. (2013), Flow shop instances, http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html.