Perturbation in Local Search for Scheduling

Professor Dr.-Ing. Frank Herrmann
Ostbayerische Technische Hochschule Regensburg
Innovation and Competence Centre for Production Logistics and Factory Planning (IPF)
PO box 120327, 93025 Regensburg, Germany
Email: Frank.Herrmann@OTH-Regensburg.de

KEYWORDS

Local Search, perturbation, scheduling, flow shop

ABSTRACT

Local search is an established meta heuristic for scheduling problems. To avoid of getting stucked in a local optimum, the achieved solution is destroyed. Usually, this is done randomly. Here, two problem specific procedures are suggested. One focuses on jobs with the highest difference between the actual processing time and the net processing time of the job and the other one tries to remove jobs which causes the highest idle time for themselves or other jobs. Comprehensive simulations of test scheduling problems as well as a real world application show that both problem specific procedures outperform a random procedure.

INTRODUCTION

Specific products are produced by special machines which are often grouped in a flow shop. Often these production systems deliver products for other systems as well. Due to the hierarchical planning which is implemented in enterprise resource planning systems (ERP system) (see e.g. Jacobs et al. 2010), the local completion times in one production system in many cases determine the earliest possible starting times in another production system. So, the delay of the operations in a production system has an impact on the effectivity of this coordination process. The tendency to a lot size of one, small batches with short response times have to be produced. Thus, scheduling algorithms are needed to ensure that under the constraint of a high average load of the flow shop, the due dates of the production orders are met. Nowadays, such special designed flow shops often have technological restrictions, which complicate the scheduling. For example in cell manufacturing, buffer could be non-existent due to limited space and storage facilities.

These high demands result in more complex solution spaces. If there is no information that indicates where an optimal solution can be found then all elements of the solution space must be evaluated and all possible optimisation algorithms show the same behaviour. Especially, random search, which iteratively generate solutions randomly, has the same performance. An example is the needle-in-a-haystack problem. In the opposite case, guided search methods which iteratively calculate solutions and use the objective values of previously determined solutions to guide the future search process, are successful. The degree of usage depends on the locality of the solution space. Due to many published variants of guided search, a sufficient locality of scheduling problems is available.

In this paper, a local search is investigated on a permutation flow shop problem. With a neighbourhood operator a new permutation is calculated and by a simulation routine the resulting tardiness is determined (via the resulting schedule). Motivated by the gradient descent methods in mathematical optimisation one step of the local search, starting with an initial permutation, terminates by reaching a better permutation. This process is iterated until the solution no longer can be improved. Usually, this leads to a local optimum. To leave this local optimum and come to another local optimum or even to reach the global optimum, a destruction is necessary. Often, destruction is done randomly. Two alternatives are investigated. One focuses on jobs with the highest difference between the actual processing time and the net processing time of the job. The second one tries to remove jobs which causes the highest idle time for themselves or other jobs. In comprehensive simulations with a real world application both alternative destructions outperform a random one.

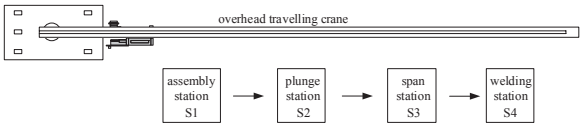
A REAL WORLD APPLICATION

The problem is a modification of a partly automated production line at Fiedler Andritz in Regensburg to produce filter (baskets) with a lot size of 1. All filters have unified constructions. They differ in varying heights of the baskets and there exist different designs.

The production line consists of 4 stations which are shown in Figure 1. Station 1 assembles 6 single batons (called consoles) on an assembly ground plate to a skeleton of a filter basket. Baton profiles are assembled into the provided slots of the filter basket skeletons. At the plunge station a wire coil is contrived in the device of a lining machine. The lining machine straightens the wire and inserts batons into the slots. To ensure stability, the span station installs span kernels in the case of outflow filter baskets and span belts in the case of inflow filter baskets. Then, the filter basket is lifted from the assembly ground plate and is transported to the welding station, at which the baton profiles are welded on the filter basket skeletons. The completed filter basket leaves the production line. Prior to this, the span medium is removed. An overhead travelling crane lifts a filter basket out of a station, transports it to the next station and inserts it directly in this station. This is just possible if this station is free. So, there is no buffer in the production

Proceedings 29th European Conference on Modelling and Simulation ©ECMS Valeri M. Mladenov, Petia Georgieva, Grisha Spasov, Galidiya Petrova (Editors)
ISBN: 978-0-9932440-0-1 / ISBN: 978-0-9932440-1-8 (CD)

Figure 1: Structure of the production line



line and each feasible schedule of jobs is a permutation of these jobs. Due to other operational issues the crane can just be moved if all stations are inactive. Since an operation cannot be interrupted, the transport has to be performed after the completion of all operations on the stations in the flow shop. Due to further operational issues this restriction has to be applied also for the first and the last station; note, that the crane loads S1 and unloads S4 as well. In summary, all stations are loaded and unloaded with filters during a common process and this process starts with the last station S4, followed by station S3, S2, until station S1 is reached. It is allowed that a station is empty; then this station is skipped (may be partially) in this process. There are 10 part types. Their different heights or designs causes different processing times which are listed in Table I.

Table I: Processing times for all part types in minutes

Part line type	Station S1	S2	S3	S4	Sum of times
P1	100,5	50	53,5	9	213
P2	256,5	50	53,5	9	369
P3	122	135	90	75	422
P4	256,5	50	267	9	582,5
P5	182	200	135,5	140	657,5
P6	100,5	300	53,5	300	754
P7	223	250	196	220	889
P8	223	250	206,5	220	899,5
P9	100,5	300	267	300	967,5
P10	256,5	300	267	300	1123,5

At the company's production site, the jobs for filters are generated by an SAP system and produced filters are stored before they are assembled into other products or sold directly to customers. Therefore, all jobs with a release date after the beginning of a period are released at the beginning of this period. One period consists of one day with three 8 hour shifts. For this investigation, sequences of jobs of filter types with lot size 1 are randomly generated for each period t by an generating algorithm which has been designed in accordance with the proceeding in (Russel et al. 1987) and in (Engell et al. 1994): An additional filter type F, released in period t, consumes capacity on each station during the time between t and its due date; the calculation for the capacity just uses the net processing time and does not regard the dependencies between the jobs (released so far). F is accepted as long as this consumed capacity on each station is below a maximal load level, otherwise it will be skipped to the next period. A maximal load level is an (intended) average load $(L_0(S))$ plus 0, 30% and +30% of $(L_0(S))$. Over the first 5, 10, 15 etc. consecutive periods, the load level variations average to zero. In reality at the company, there are large numbers of periods with a low number of late jobs and large numbers of periods with a high (or even very high) number of late jobs. To achieve a comparable situation for this investigations, due dates are determined in a way so that scheduling with the FIFO rule (first-in-first-out) causes a specific percentage of late jobs. The company confirmed that job sequences with 30%, 50%, 70% and 85% of late jobs by scheduling with the FIFO rule (called time pressure) are comparable to the ones which occurred in the real operation. As a result of the generating algorithm's calculations the mean difference between the due dates and the release dates are between 2 and 3,5 days with a standard deviation of 0,5 days. Andritz Fiedler has confirmed that such timeframes for processing jobs are representative.

The time needed for loading and unloading a station is negligible compared to the duration of the operation itself. In addition this task is independent from the allocation (or loading) of other stations and the required time is included in the duration of the operation.

The general scheduling problem consists of M stations and a pool of N jobs, which may change at any time, with known earliest possible starting times (or release dates) a_i $(1 \leq i \leq N)$ and due dates f_i $(1 \leq i \leq N)$ respectively. Also there is the duration $t_{i,j}$ of operation j of job i; i.e. $o_{i,j}$ $(1 \leq j \leq M,\ 1 \leq i \leq N)$, which is being worked on station j. Due to the reasons, said in the introduction, as performance criteria average tardiness (T_{mean}) and standard deviation of the tardiness (T_σ) are primarily analysed.

The time between two consecutive executions of the load process is determined by the maximum of the duration of the operations (including setup time) on the stations in the flow shop. This is called cycle time. This load-restriction, the no-buffer condition and the capacity of the stations are the main restrictions.

The no-buffer condition means a relaxation of the scheduling problem with the (above) load-restriction. Scheduling problems with the no-buffer are proven to be NP-hard in the strong sense for more than two stations; see e.g. (Hall and Sriskandarajah 1996), which contains a good survey of such problems.

LITERATURE REVIEW

As mentioned earlier, the real application is close to the class of no-buffer (blocking) scheduling problems. Solutions for the no-buffer (blocking) scheduling problems are published in various papers. Most papers minimise makespan as the ones in the following review — afterwards publications to minimise tardiness are reviewed. Thus, the following review explains a large spectrum of possible procedures. In (McCormick et al. 1989) a schedule is extended by a (unscheduled) job that leads to the minimum sum of blocking times on machines which is called profile fitting (PF). Often the starting point of an algorithm is the NEH algorithm presented in (Nawaz et al. 1983), as it is the best constructive heuristic to minimize the

makespan in the flow shop with blocking according to many papers, e.g. (Framinan et al. 2003). Therefore, (Ronconi 2004) substituted the initial solution for the enumeration procedure of the NEH algorithm by a heuristic based on a makespan property proven in (Ronconi and Armentano 2001) as well as by the profile fitting (PF) of (McCormick et al. 1989). (Ronconi 2005) used an elaborated lower bound to realise a branch-and-bound algorithm which becomes a heuristic since the CPU time of a run is limited. Also for minimising makespan, (Grabowski and Pempera 2007) realised and analysed a tabu search algorithm. As an alternative approach, (Wang and Tang 2012) have developed a discrete particle swarm optimisation algorithm. In order to diversify the population, a random velocity perturbation for each particle is integrated according to a probability controlled by the diversity of the current population. Again, based on the NEH algorithm, (Wang et al. 2011) described a harmony search algorithm. First, the jobs (i.e. a harmony vector) are ordered by their non-increasing position value in the harmony vector, called largest position value, to obtain a job permutation. A new NEH heuristic is developed on the reasonable premise that the jobs with less total processing times should be given higher priorities for the blocking flow shop scheduling with makespan criterion. This leads to an initial solution with higher quality. With special settings as a result of the mechanism of a harmony search algorithm, better results are achieved. Also (Ribas et al. 2011) presented an improved NEH-based heuristic and uses this as the initial solution procedure for their iterated greedy algorithm. A modified simulated annealing algorithm with a local search procedure is proposed by (Wang et al. 2012). For this, an approximate solution is generated using a priority rule specific to the nature of the blocking and a variant of the NEH-insert procedure. Again, based on the profile fitting (PF) approach of (McCormick et al. 1989), (Pan and Wang 2012) addressed two simple constructive heuristics. Then, both heuristics and the profile fitting are combined with the NEH heuristic to three improved constructive heuristics. Their solutions are further improved by an insertion-based local search method. The resulting three composite heuristics are tested on the well-known flow shop benchmark of (Taillard 1993), which is widely used as benchmark in the literature.

To the best of my knowledge, only a few studies investigate algorithms for the total tardiness objective (for flow shops with blocking). (Ronconi and Armentano 2001) have developed a lower bound which reduces the number of nodes in a branch-and-bound algorithm significantly. (Ronconi and Henriques 2009) described several versions of a local search. First, with the NEH algorithm, they explore specific characteristics of the problem. A more comprehensive local search is developed by a GRASP based (greedy randomized adaptive search procedure) search heuristic. There are just a few genetic algorithms with this performance criteria: for a no-wait flowshop scheduling problem one is published in (Chaudhry and Mahmood 2012) and for a flowshop with blocking one is published in (Januario et al. 2009). There are several local search algorithms for the scheduling problem in general and in particular for the flow shop problem. For an overview see (Ruiz and Maroto 2005) or (Pan and Ruiz 2012).

LOCAL SEARCH

All guided search procedures bases on a metric of the search space consisting of all feasible solutions - here, all permutations of all orders. Such a metric measures the similarity of solutions by a distance which implies a neighbourhood structure, s. e.g. (Burke and Kendall 2014). Several metrics are suggested, s. e.g. (Rothlauf 2011) or (Burke and Kendall 2014). Especially for binary search spaces (i.e. each position in a vector has a binary value), the Hamming metric is used, s. e.g. (Rothlauf 2011) or (Burke and Kendall 2014). The Hamming metric counts the number m of different positions of two permutations and m is then the Hamming distance between these two permutations. If there is a Hamming distance of $(n + 1)$ between two permutations p and q then there is a sequence s of n permutations between of two positions, such that s transforms p in q. In this paper an iteration of (such) permutations between of two positions is a local search; thus a sequence of direct neighbours (due to their Hamming distance of 2) is created.

The local search algorithm, formally described in Algorithm 1, creates gradually random neighbours, starting from an initial permutation p^* (=p, respectively) (for the representation of a schedule). The current permutation is replaced by a neighbour who by all neighbours has the best rating (calculated by the objective function value f) and its rating is better than the current permutation; this is called a "best fit". The result is either a better permutation or the current permutation could not be improved. This is referred to as a step in the local search. If an improvement has been achieved, then the local search step is reapplied.

If an iteration returns no improvement, it does not mean that any further application of the local search step does not lead to an improvement. Such an improvement is possible if the orders are tried in a different order. Tests show that often just marginal improvements are achieved. Due to the consumed (computation) time – which can be used more efficiently as will be explained below the local search algorithm terminates if an iteration does not cause an improvement; this is in line with (Pan and Ruiz 2012).

To determine the initial permutation various methods can be used. The permutation can for example be fixed or generated by a meta-heuristic. In (Ruiz and Stützle 2007) for example, the NEH heuristic (Nawaz et al 1983) is proposed. According to (Adenso-Díaz 1992) the quality of the initial permutation can have a significant impact on the computation time of an algorithm and save up to 10% of the computation time.

Often the local search algorithm terminates in a local optimum. To find a global optimum, this local search algorithm is restarted. Possible restarts are other initial permutations or a change of this local optimum, called perturbation, which is the preferred procedure in the literature, s. e.g. (Ruiz and Maroto 2005) and (Pan and Ruiz 2012). Typically, the application of the neighbours of a local optimum (o) terminates in o again;

Algorithm 1 local search algorithm

initial permutation p^*
do
  $p^{start} = p^*$
  $p = p^*$
  for $i = 1$ to size of p **do**
    for $j = i + 1$ to size of p **do**
      p^t by changing orders in positions i and j in p
      if $f(p^t) < f(p^*)$ **then**
        $p^* = p^t$
      end if
    end for
  end for
while $f(p^*) < f(p^{start})$

Some comments: p^{start} is the temporary best permutation for one step of the local search (in Do-loop), p^* is the temporary best permutation for direct neighbours (in For-loop) and p is the permutation whose direct neighbours are calculated (in For-loop)

Algorithm 2 iterative local search algorithm

initial permutation p
temp best permutation $p^* = p$
apply local search on p with result p
for $i = 1$ to number of iterations **do**
  $p =$ perturbation(p)
  apply local search on p
  if $ofv(p) < ofv(p^*)$ **then**
    $p^* = p$
  end if
end for

comment: as perturbation use random, algorithm 3 or algorithm 4.

this is called basin of attraction. Therefore, a perturbation of an optimum (o) must be so significant that the basin of attraction is left. This sufficient condition is fulfilled by a large perturbation. On the other hand, it is expected that the local optimum contains parts of an optimal solution. These favourable parts should be maintained. Own tests with several scheduling problems show that this sufficient condition is often reached by direct neighbours of a local optimum. Such a perturbation basically extends the local search algorithm to the iterative one by applying a perturbation on the result of one run of the local search algorithm and this is iterated; formally described in Algorithm 2. In literature a random baesed perturbation mechanism (random destruction - RD) is preferred; s. e.g. (Ruiz and Maroto 2005) and (Pan and Ruiz 2012) and (Pan and Wang 2012). It removes γ orders from the (initial) permutation π, which causes a sequence π' of length $\pi - \gamma$ and a set π_r of the removed γ orders. Then, the orders of the set π_r will be re-inserted at random positions in the sequence π'. As an alternative the perturbation is controlled by the difference between the actual throughput time and the net

Algorithm 3 processing time perturbation - PT

permutation p
empty order list ol
for $i = 1$ to perturbation-size **do**
  determine order o by max(actual throughput time$-$net processing time)from p
  delete o in p
  insert o in ol
end for
insert all orders from ol in p at random positions

Algorithm 4 idle time perturbation - ID

permutation p
cycle list cl generated out of p
empty order list ol
for $i = 1$ to perturbation-size **do**
  determine cycle c with highest idle time from cl
  determine first order o in c
  delete c in cl
  delete o in p
  insert o in ol
end for
insert all orders from ol in p at random positions

processing time of an order and by the cycles for the special production type. In the first approach the orders are ordered in descending order by this difference and the first γ orders are removed; formally this is described in Algorithm 3 (processing time perturbation - PT). The second approach tries to remove jobs which causes the highest idle time for themselves or other jobs. Thus, the orders are ordered in descending order by the hight of the idle time caused by this order and the first γ orders are removed; formally this is described in Algorithm 4 (idle time perturbation - ID). Note: ID addresses the restriction that the transportation of the work pieces only can be done by an overhead crane.

COMPUTATIONAL RESULTS

In order to use problems, which are comparable to the real world problem, 4 stations are being considered. The routings are created from a set of so called basis routings which are stated in table 2. The total net processing times of these basis routings covers the same range as the ones in the real world application. A concrete routing is created from one basis routing R, as routing 3 for example. The processing time for a station S, as station 2 for example, is created by a normal distribution whose mean value is the processing of S in R, also 200 minutes in the example, and the deviation is either 20% of the mean value (small deviation) or 75% of the mean value (large deviation); of course negative processing times are excluded – in both cases. The pool of orders is too small to effectively generate a part type sequence by the generating algorithm. Instead, the part types are generated by a uniform distribution, and the following three basic scenarios for the

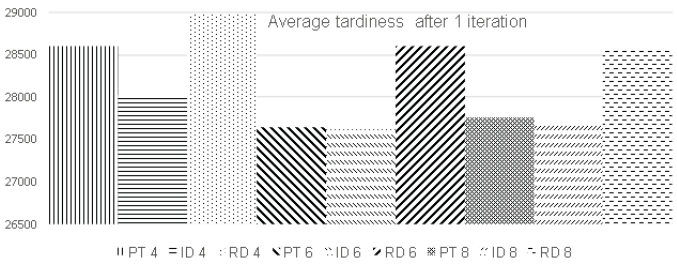
order release are randomly generated: all orders are realised in one, two or three periods, so that in each scenario the difference of the number of orders in the periods is at most one note, that sometimes orders of previous periods are still in the production system. The number of orders vary between 8, 16 and 24.
The due dates were generated by a fix flow factor, so that under scheduling with FIFO the percentage of late jobs is 30%, 50%, 70% or 85% – resulting in $3 \times 4 \times 3 = 16$ combinations. For each operation in the five basis routings (see table II) 4 processing times are generated. This leads to $5 \times 4 \times = 80$ routings. So, in total $36 \times 80 = 2880$ experiments are generated.

Table II: Basis routings for the creation of scheduling problems in minutes

Routing / part line type	Station 1	Station 2	Station 3	Station 4
1	100	50	50	10
2	150	100	100	150
3	100	200	150	200
4	200	150	300	150
5	250	250	250	200
Sum of operation times	800	750	850	710

By using the basis routings and a uniform distribution of the parts, station 3 is the bottleneck station, because the sum of all operation times at station 3 is greater than these sums at the other stations. Because of the way alternative processing times are being generated for the stations, the sequence of stations due to this criterion (the sum of all operation times at a station) can change. In the real application each station is a bottleneck in a significant portion of the periods over a large horizon, because the products are non-uniform distributed in the demand over a large horizon. In order to ensure a comparable situation, about 60000 are generated. From those 2880 are chosen, so that in around 15% of the experiments station 1 is a bottleneck station, in around 30% of the experiments station 2 is a bottleneck station, in around 25% of the experiments station 3 is a bottleneck station, and in around 30% of the experiments station 4 is a bottleneck station (and of course, the other conditions are still fulfilled).
The effectivity of these three perturbation methods is analysed by improving the average tardiness in the local search environment. For this, in a test step, a perturbation method is applied on a local optimum and, then, on the result the local search algorithm is applied. Local optimas are generated by applying the local search algorithm on 10 initial permutations. Three of them are calculated by the priority rules shortest processing time, earliest due date and slack. With the above explained 2880 test problems there are $2880 \cdot 10$ test steps. 5000 of them are chosen randomly.
In a preliminary investigation the size of the perturbation, i.e. parameter γ, is analysed. Reducing γ increases the probability of the same result by the next run of a local search algorithm. An increase of γ increase the random choice, thus, in the extreme case, processing time perturbation as well as idle time

Figure 2: average tardiness after one perturbation with different parameters and then an application of the local search algorithm



perturbation are random destructions. One and the maximum number of orders determines the possible setting for γ; as said earlier, the orders vary between 8, 16 and 24. Each combination of parameter γ with each kind of perturbation, i.e. processing time perturbation (PT), idle time perturbation (ID), and random perturbation (RD) is used in 20 test steps; which are randomly chosen out of the above 5000 test steps. The results with γ of 4, 6, and 8 outperform the others by a factor of at least as 2 – often even larger. And the differences in the results between these settings of γ are significant smaller. Each combination of the most successful parameter setting of γ, i.e. 4, 6, and 8, with each kind of perturbation, i.e. processing time perturbation (PT), idle time perturbation (ID), and random perturbation (RD) is applied on the above 5000 test steps. The resulting average tardiness is in Figure 2. It shows that the value of 4 for γ is very unfavourable. Random perturbation is outperformed by both alternatives. But, for some test steps there is no considerable difference. Nevertheless, this demonstrates the advantage of a problem specific perturbation. The advantage of the idle time perturbation shows that the transportation limitation by the overhead travelling crane in the real world application has a significant impact on the scheduling. The small difference between the idle time perturbation and the processing time perturbation shows the effectivity of the processing time perturbation. Thus, substituting random perturbation by processing time perturbation should be beneficial in the general case. Experiments with this test problem without the transportation limitation demonstrate this impressively.

For this parameter setting the speed of convergence is analyzed for a problem instance with a percentage of late jobs of 70% by using the FIFO priority rule. Starting from a randomly created initial permutation local search is repeated 50 times successively. Compared to random perturbation both problem specific ones found the best solution earlier ans they delivered better results in case of a short allowed runtime. Both effects might occur in general.
The real world application is simulated for the sequence of orders explained in the section about the real world application. If an assignment of an operation on the first station ends in the next period t, the orders of period (t+1) is realised, so

that the orders of the periods t and (t+1) are known. Average tardiness (T_{mean}) and standard deviation of the tardiness (T_σ) reach a steady state by a simulation horizon of 5000 periods. Due to the results published in Herrmann 2013 and Herrmann 2014 local search is compared with the results of the best priority rules, which is RR, and a genetic algorithm (GA). With total processing time of job i (tt_i) is $RR = (f_i - t - tt_i) \cdot e^{-\eta} + e^{\eta} \cdot tt_i$, with utilisation level η of the entire flow shop defined by $\eta = \frac{b}{b+j}$ with b being the busy time and j being the idle time of the entire flow shop. For local search the best two settings for γ, i.e. 6 and 8, are regarded for all three perturbation methods (abbreviated by: LS for local search followed by the perturbation abbreviation used before and setting of γ in parenthesis; example: LSID(6) means local search with idle time perturbation and ($\gamma = 8$)). The results shown in table III are average objective values relative to the solutions of GA set to 100%; thus, e.g., the solutions generated by RR rule for time pressure 30% were about 51% above GA on the average.

Table III: Basis routings for the creation of scheduling problems in minutes

Procedure	Time pressure			
T_{mean}	30%	50%	70%	85%
GA	100	100	100	100
RR	1,51	1,41	1,33	1,21
LSRD(6)	0,97	0,97	0,94	0,95
LSRD(8)	0,96	0,95	0,96	0,97
LSPT(6)	0,91	0,93	0,91	0,93
LSPT(8)	0,88	0,89	0,87	0,90
LSID(6)	0,89	0,90	0,89	0,85
LSID(8)	0,83	0,89	0,87	0,83
T_σ	30%	50%	70%	85%
GA	100	100	100	100
RR	1,21	1,19	1,13	1,09
LSRD(6)	0,94	0,97	0,98	0,93
LSRD(8)	0,96	1,02	0,97	0,96
LSPT(6)	0,88	0,91	0,95	0,98
LSPT(8)	0,90	0,89	0,93	0,91
LSID(6)	0,86	0,87	0,91	0,96
LSID(8)	0,85	0,86	0,89	0,89

The genetic algorithm outperforms the priority rules significantly. At the company site the benefit would even much higher, because their scheduling procedure is much simpler (than these priority rules). Local search with random perturbation delivers slightly better results than the genetic algorithm. The improvements by the other two problem specific perturbation methods are inline with the ones for the created scheduling problems. In total a noticeable improvement is achieved. A first analysis of optimal schedules of the above test problems show that, for example, outliers in the cycle times are avoided. Idle perturbation avoids this more often than random perturbation.

CONCLUSIONS

This paper analyses two problem specific perturbation procedures leaving a local optimum in a local search procedure for flow shop permutation problems with a significant technical restriction occurred in a real world application with more restrictive restrictions than the scheduling problems normally regarded in the literature. The standard perturbation is outperformed. A further improvement seems to be possible by integrating specific properties of very good schedules, especially in the operators. More technical restrictions in companies occur by limited resources, like the available number of coils or assembly ground plates, and workers for the manual tasks causes other schedules to be optimal or at least very good. Such requirements are also left to future investigations.

REFERENCES

Adenso-Díaz, B. 1992. "Restricted neighbourhood in the tabu search for the flowshop problem. European Journal of Operational Research 62, 27 – 37.

Burke, E. and G. Kendall. 2014. "Search Methodologies: Introductory tutorials in optimization and decision support techniques". Springer, Boston, MA, USA.

Januario, T.; J. Arroyo and M. Moreira. 2009. Genetic Algorithm for Tardiness Minimization in Flowshop with Blocking. In: N. Krasnogor, B. Melin, J. Moreno, J. Moreno-Vega, D. Pelta (Editors): Nature Inspired Cooperative Strategies for Optimization (NICSO 2008). Studies in Computational Intelligence Volume 236, 2009, 153 - 164.

El-Bouri, A. 2012. A cooperative dispatching approach for minimizing mean tardiness in a dynamic flowshop. Computers & Operations Research, Volume 39, Issue 7 (July), 1305 - 1314.

Chaudhry, I.; S. Mahmood. 2012. No-wait Flowshop Scheduling Using Genetic Algorithm. In Proceedings of the World Congress on Engineering, Vol III, WCE 2012, July 4 – 6, 2012, London, U.K..

Dong, X.; Huang, H.; and Ping, C. 2009. "An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion". Computers & Operations Research Volume 36, 1664 - 1669. Engell, S.; F. Herrmann; and M. Moser. 1994. Priority rules and predictive control algorithms for on-line scheduling of FMS. In Computer Control of Flexible Manufacturing Systems, S.B. Joshi and J.S. Smith (Eds.). Chapman & Hall, London, 75 - 107.

Framinan, J.M.; R. Leisten; and C. Rajendran. 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, 121 - 148.

Grabowski, J. and J. Pempera. 2007. The permutation flow shop problem with blocking. A tabu search approach.. Omega, 35 (3), 302 - 311.

Hall, N.G. and C. Sriskandarajah. 1996. A survey of machine scheduling problems with blocking and no-wait in process. Operations Research 44 (3), 510 - 525.

Herrmann, F. 2013. Simulation based priority rules for scheduling of a flow shop with simultaneously loaded stations. In: Proceedings of the 27th EUROPEAN Conference on Modeling and Simulation, May 27th - 30th, 2013, lesund, Norway.

Herrmann, F. 2014. Genetic algorithm with simulation for scheduling of a flow shop with simultaneously loaded stations. In: Proceedings of the 28th EUROPEAN Conference on Modeling and Simulation, May 27th - 30th, 2014, Brescia, Italy.

Jacobs, F.R.; W. Berry; D. Whybark; T. Vollmann. 2010. Manufacturing Planning and Control for Supply Chain Management. McGraw-Hill/Irwin (New York), 6 edition.

Lawrence, S. and T. Morton. 1993. Resource-constrained multi-project scheduling with tardy costs: Comparing myopic, bottleneck, and resource pricing heuristics.. European Journal of Operational Research 64, 168 - 187.

McCormick, S.T.; M.L. Pinedo; S. Shenker; and B. Wolf. 1989. Sequencing in an assembly line with blocking to minimize cycle time. Operations Research, 37 (6), 925 - 935.

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

Pan, Q.; and L. Wang. 2012. Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. Omega, 40 (2), 218 - 229.

Pan, Q, and R. Ruiz. 2012. "Local search methods for the flowshop scheduling problem with flowtime minimization. European Journal of Operational Research, 222, 31 - 43.

Rachamadugu, R.M.V. 1987. Technical Note A Note on the Weighted Tardiness Problem. Operations Research, 35, 450 452.

Rachamadugu, R.V. and T.E. Morton. 1982. Myopic heuristics for the single machine weighted tardiness problem. Working Paper No. 28-81-82, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA.

Raghu, T.S. and C. Rajendran. 1993. An efficient dynamic dispatching rule for scheduling in a job shop. International Journal of Production Economics 32, 301 - 313.

Rajendran, C and O. Holthaus. 1999. A comparative study of dispatching rules in dynamic flowshops and job shops. European Journal of Operational Research, 116 (1), 156 - 170.

Ribas, I.; R. Companys; and X. Tort-Martorell. 2011. An iterated greedy algorithm for the flowshop scheduling problem with blocking. Omega, 39, 293 - 301.

Ronconi, D.P. and V.A. Armentano. 2001. Lower Bounding Schemes for Flowshops with Blocking In-Process. Journal of the Operational Research Society, 52 (11), 1289 - 1297.

Ronconi, D.P. 2004. A note on constructive heuristics for the flow-shop problem with blocking. International Journal of Production Economics, 39 - 48.

Ronconi, D.P. 2005. A branch-and-bound algorithm to minimize the makespan in a flowshop with blocking. Annals of Operations Research, (138), 53 - 65.

Ronconi, D. and L. Henrique. 2009. Some heuristic algorithms for total tardiness minimization in a flow shop with blocking. Omega, 37 (2), 272 - 281.

Rothlauf, Franz. 2011. "Design of Modern Heuristics: Principles and Application". Springer, Berlin, Germany.

Russel, R.S.; E.M. Dar-El; and B.W. Taylor. 1987. A comparative analysis of the COVERT job sequencing rule using various shop performance measures. International Journal of Production Research, 25 (10), 1523 - 1540.

Ruiz, R. and T. Stützle 2007. "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem". European Journal of Ruiz, R. and C. Maroto 2005. "A comprehensive review and evaluation of permutation flowshop heuristics". European Journal of Operational Research, 165, 479 - 494.

Taillard, E. 1993. Benchmarks for basic scheduling problems. European Journal of Operational Research, 64 (2), 278 285.

Vepsatainen, A.P. and T.E. Morton. 1987. Priority rules for job shops with weighted tardiness costs. Management Science 33/8, 95 - 103.

Vo, S. and A. Witt. 2007. Hybrid Flow Shop Scheduling as a Multi-Mode Multi-Project Scheduling Problem with Batching Requirements: A real-world application.. International Journal of Production Economics 105, 445 - 458.

Wang, X. and L. Tang. 2012. A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flow shop problem with blocking. Applied Soft Computing, (12, 2), 652 - 662.

Wang, L.; Q.-K. Pan; and M.F. Tasgetiren. 2011. A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem. Computers & Industrial Engineering, 61 (1), 76 - 83.

Wang, C.; S. Song, S.; J.N.D. Gupta; and C. Wu. 2012. A three-phase algorithm for flowshop scheduling with blocking to minimize makespan. Computers & Operations Research, 39 (11), 2880 - 2887.

Werner, F. 2013. A survey of genetic algorithms for shop scheduling problems. In: Heuristics. - New York, Nova Publishers, 133 - 160.

AUTHOR BIOGRAPHY

Frank Herrmann

was born in Münster, Germany and went to the RWTH Aachen, where he studied computer science and obtained his degree in 1989. During his time with the Fraunhofer Institute IITB in Karlsruhe he obtained his PhD in 1996 about scheduling problems. From 1996 until 2003 he worked for SAP AG on various jobs, at the last as director. In 2003 he became Professor for Production Logistics at the University of Applied Sciences in Regensburg. His research topics are planning algorithms and simulation for operative production planning and control. His e-mail address is Frank.Herrmann@OTH-Regensburg.de and his Web-page can be found at www.hs-regensburg.de/frank.herrmann.