*

Rescheduling in Job-Shop Problems for Sustainable Manufacturing Systems

Miguel A. Salido, Joan Escamilla, Federico Barber, Adriana Giret
Instituto de Automáica e Informáica Industrial
Universitad Politècnica de València, Spain

in: Journal of Cleaner Production 162.(2017) http://dx.doi.org/10.1016/j.jclepro.2016.11.002

Abstract

Manufacturing industries are faced with environmental challenges so their industrial processes must be optimized in terms of both profitability and sustainability. Most of these processes are dynamic, so the previously obtained solutions cannot be valid after incidences or disruptions. This paper is focused on recovery in dynamic job-shop scheduling problems where machines can work at different rates. Machine speed scaling is an alternative framework to the on/off con-trol framework for production scheduling. Thus, given a disruption, the main goal is to recover the original solution by rescheduling the minimum number of tasks. To this end, a new match-up technique is developed to determine the rescheduling zone and a feasible reschedule. Then, a memetic algorithm is pro-posed for finding a schedule that minimize the energy consumption within the rescheduling zone but maintaining the makespan constraint. An extensive study is carried out to analyze the behavior of our algorithms to recover the original solution and minimize the energy reduction in different benchmarks, taken from the OR-Library. The energy consumption and processing time of the involved tasks in the rescheduling zone will play an important role to determine the best match-up point and the optimized rescheduling. Upon a disruption, different rescheduling solutions can be obtained, all of them holding with the require-ments, but with different values of energy consumption. The results proposed in this paper may be useful to be applied in real industries for energy-efficient production rescheduling.

Keywords: Manufacturing problem, Multi-objective, Rescheduling, Memetic algorithm, Energy consumption

1. Introduction

In manufacturing industries, there exist many unexpected disruptions every day (machine breakdown, order modification, disruptive events, order cancelations, etc). After a disruption, the original schedule may become invalid by the new conditions. In some cases, it is possible to easily modify the solution to absorb the disruption but in many cases rescheduling is mandatory to minimize the effects of such disruption and recover the original solution as soon as possible.

In the literature, there are many dynamic scheduling methods to manage online scheduling. Arnaout (2014) tackles rescheduling for the unrelated paral-lel machine problem with sequence dependent setup times and different rates of breakdowns or urgent jobs arrivals. To this end, a new repair rule referred to as Minimum Weighted Cmax Difference (MWCD) is developed and compared to existing algorithms based on both schedule quality and stability. Hall and Potts (2004) work with scheduling problems where a set of original jobs has already been scheduled to minimize some cost objective when a new set of jobs arrives and creates a disruption. The decision maker needs to insert the new jobs into the existing schedule without excessively disrupting. The authors pro-vide either an efficient algorithm or a proof that such an algorithm is unlikely to exist. In Qi et al. (2006) the problem of updating a machine schedule is pro-posed when either a random or an anticipated disruption occurs after a subset of the jobs. The proposed approach differs from most rescheduling analysis in that the cost associated with the deviation between the original and the new schedule is included in the model. Vieira et al. (2000) presents new analytical models that can predict the performance of rescheduling strategies and quan-tify the trade-off between different performance measures. To this end, three rescheduling strategies are studied: periodic, hybrid, and event-driven based

on the queue size. Vieira et al. (2003) present a framework for understanding rescheduling strategies, policies and methods in rescheduling manufacturing systems. The work explains methods for generating robust schedules and methods for updating schedules. In Subramaniam and Raheja (2003), the typical job shop disruptions are studied and their repair processes are decomposed into four generic repair steps, which are achieved using a modified affected operation rescheduling (mAOR) heuristic. In Herroelen and Leus (2004), several methodologies for proactive and reactive project scheduling are reviewed. They also offer a framework that allow project management to identify the proper scheduling methodology for different project scheduling environments.

Furthermore, the main objective of manufacturing industries is to improve profitability and competitiveness. These improvements can be obtained with a good optimization of resources allocation. In the last years, many industries are not only facing complex and diverse economic trends of shorter product life cycles, quick changing science and technology, increasing customer demand diversity, and production activities globalization but also enormous and heavy environmental challenges of global climate change (Mestl et al. (2005)) and rapid exhaustion of various non-renewable resources (Yusoff (2006)). Research on reducing the energy consumption of manufacturing processes has mainly focused on the energy consumption optimization based on the machine level and the product level (Neugebauer et al. (2011)). In Gahm et al. (2016), a research framework for energy-efficient scheduling is developed. Different proposals have been classified following different attributes and criteria. Tonelli et al. (2016) propose a centralized and distributed model for an off-line energy-aware scheduling problem. Liu et al. (2014) propose a model for the bi-objectives problem that minimizes total electricity consumption and total weighted tardiness in JSP. To this end the Non-dominant Sorting Genetic Algorithm is employed to obtain the Pareto front. In May et al. (2015), a green genetic algorithm is proposed to achieve a semi-optimal makespan with a significantly lower total energy consumption in job shop scheduling problems. The study demonstrated that the worthless energy consumption can be reduced significantly by employing com-

plex energy-efficient machine behavior policies. Mouzon et al. (2007) developed several algorithms and a multiple-objective mathematical programming model to investigate the problem of scheduling jobs on a single CNC machine in order to reduce energy consumption and total completion time. They pointed out that there was a significant amount of energy savings when non-bottleneck machines were turned off until needed. It is well-known that machines consume a considerable amount of energy when left idle. Thus, many works propose a turn-on and turn-off scheduling framework to control the machines. Thus the overall energy consumption can be reduced. For some manufacturing systems, however, it is not possible to turn off machines completely during each of the idle intervals, either because restarting the machines requires a large amount of energy or because frequent on and off switches may damage to the machine components. In these cases, the on/off control framework is not applicable (Zhang and Chiong (2016)). Thus, an alternative to the on/off control framework is a new framework based on machine speed scaling (Fang et al., 2013 for flow shop scheduling)(Salido et al. 2013 for job shop scheduling). In this new framework, machines are allowed to work at different speed levels when processing different jobs. Some researchers have focused their research in this framework. In Zhang and Chiong (2016), a multiobjective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption is proposed. Fang et al. (2013) propose mathematical programming and combinatorial approaches to consider a flow shop scheduling problem with a restriction on peak power consumption.

One of the most important production scheduling problems studied in the literature is the job-shop scheduling problem (JSP) that represents a problem in which some tasks are assigned to machines with a specific processing time. In comparison, studies on the JSP with energy-saving objectives are limited, although currently some works are considering this feature in JSPs.

This paper works with an extension of the job-shop scheduling problem where each machine can work at different rates (JSMS) (Salido et al. (2013)). It is assumed that when a job is processed at a higher speed, its processing time

decreases, while its power consumption increases (Fang et al. (2013)). Power consumption is the energy consumption per unit of time. Thus the energy consumption of a task with duration time is given by the formula Energy = Power * Time. In tasks related with manufacturing processes is usually assumed that as higher power in machines, less processing time is required. This relationship is not lineal since it depends on the efficient rate, which usually decreases from a certain point of the power supplied Draganescu et al. (2003). Thus, although the power consumption increases, the energy consumption may be lower, equal or higher depending on the point of the efficiency rate where machine is working. For instance, if all machines can use less energy at maximum speed for all tasks, the problem remains trivial with respect to power consumption, due to the fact that only this machine speed (from available) will be selected. Thus, the problem remains a classical job shop scheduling with the only objective of minimizing makespan. Indeed if a machine uses less energy at maximum speed (and therefore lowest processing time), the other available machine speeds can be removed from the list, in a preprocess step, due to the fact that they will not take part of a solution. Thus, it is considered the case in which there is a tradeoff between energy consumption and machine speed so all available machine speeds can take part of a solution according to operator preferences.

Thus, without loss of generality, it is considered energy consumption instead of power consumption, as the processing time to execute a task at each machine speed is known in advance. Similar to Zhang and Chiong (2016), the processing time in JSMS depends of the machines speed and therefore the energy consumption. Thus, it is assumed that increasing the machine speed will lead to higher energy consumption despite the shorter processing time.

Furthermore, most of the existing research on reducing energy consumption in JSP has focused on static scheduling models (Zhang and Chiong (2016); May et al. (2015); Liu et al. (2014)). Thus, it is needed to develop new techniques to address rescheduling and reduce the energy consumption in job-shop scheduling problems. In this paper a new rescheduling technique is developed to recover the original solution by minimizing energy-consumption within the rescheduling

zone.

2. Rescheduling and Recovery

As it has been pointed out, unpredictable events/disruptions occur everyday in manufacturing industries. Sometimes these events can be absorbed by the original schedule and no rescheduling technique is needed. In this case, the schedule is considered robust and it is able to absorb minor disruptions. However, when the event cannot be absorbed by the schedule, a rescheduling process is required to obtain a new valid schedule. In this process, the number of affected tasks should be minimized. To this end, the concept of nervousness is used to measure the amount of changes needed to recover an original schedule. The term of nervousness was coined by Steele (1975) to be used in the context of Material Requirement Planning System. It was used by Pujawan (2004) in manufacturing problems to represent the propagation of changes at the master production schedule into instability in the requirements of parts or components at lower levels of the product structure. To adapt the definitions of robustness, stability and recoverability given in Barber and Salido (2015), it is considered:

- A schedule is ***robust*** if it is able to absorb the disruption without affecting further tasks. Thus, if a machine is disrupted during a task execution, only the end time of this task is affected by the disruption.

- A schedule is ***stable*** if there exist a new feasible schedule in the neighborhood of the disrupted schedule. Thus, if a machine is disrupted during a task execution, only the start time of few tasks along the schedule are affected by the disruption.

- A schedule is ***recoverable*** if only some few consecutive tasks are affected and the original schedule is recovered from a time point. Thus, if a machine is disrupted during a task execution, only the start time of some few consecutive tasks are affected by the disruption and the rest of the following tasks remain unaltered.

The main difference between stability and recoverability is that in recoverability, the tasks to be repaired are consecutive over time and distributed among machines. Thus, there is a time point called *match-up point*, where the original schedule is recovered and all these tasks maintain their original start time. However, in stability, the start time of tasks to be repaired may be sparse along the schedule.

During the rest of the paper, the term of recoverable schedule will be used to measure the number of changes needed from the disruption point to recover the original schedule.

The three most used schedule repair methods to recover the original schedule are: regeneration, partial rescheduling, and right-shift scheduling (Vieira et al. (2003)):

- ***Regeneration*** (Church and Uzsoy (1992), Wu et al. (1993)) constructs a complete schedule by rescheduling all the tasks. It is also called total rescheduling. This strategy takes more computational effort to run since more tasks must be scheduled. It produces the most schedule nervousness and least stability.

- ***Partial rescheduling*** (Li et al. (1993), Wu and Li (1995)) takes into account only the tasks that were affected by the incidence. This reduces the schedule nervousness and increases stability.

- ***The right-shift method*** (Abumaizar and Svestka (1997)) postpones the remaining tasks by the amount of downtime. In some cases, right-shift might be a special case of partial rescheduling. The right-shift method produces the least schedule nervousness and most schedule stability. This idea was used in Akturk and Gorgulu (1999) for determining the match-up point to reschedule as few tasks as possible.

In this paper, we focus our attention in rescheduling using a match-up technique to reduce nervousness, increase stability and recover the original schedule as soon as possible. Thus, after a machine breakdown, a match-up point for

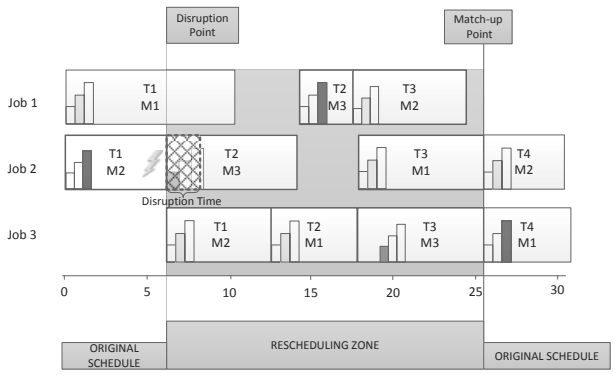

Figure 1: Rescheduling by recovery.

each machine is determined and part of the original schedule ranged between the disruption point and match-up point must be rescheduled (see Figure 1). There are two problems that must be addressed:

1. The problem of finding a match-up point for the schedule to determine the interval to be rescheduled (rescheduling zone in Figure 1). A specific technique must be developed to find the match-up point and as a result an initial and valid schedule is obtained.
2. The problem of searching for a new schedule that minimizes the energy consumption in the range between the disruption point and the obtained match-up point.

Both problems must be managed in a different way:

- The first problem could be considered a new scheduling problem where the objective is to minimize makespan (match-up point) by penalizing the modified variables with respect to the original solution. However, it could return a stable schedule and not a recovered schedule. Thus, a breadth first search technique must be developed to minimize the propagation of the disruption along the schedule.
- The second problem is a scheduling problem with a given makespan threshold, so a metaheuristic search technique must be applied/developed to minimize energy consumption. It must be taken into account that the makespan threshold is only a hard constraint in this problem and not a parameter to optimize, because the rest of the original schedule (from the match-up point) is not modified. Minimizing the makespan in this rescheduling problem could return a non-energy efficiency reschedule.

3. Problem Description

Most manufacturing industrial processes can be represented as a job-shop scheduling problem where machines can work at different speeds/rates (JSMS).

This problem consists of a set of n jobs $\{J_1, \ldots, J_n\}$ and a set of m machines $\{R_1, \ldots, R_m\}$. Each job J_i consists of a sequence of v_i tasks $(\theta_{i1}, \ldots, \theta_{iv_i})$. Each task θ_{il} has a single machine requirement $R_{\theta_{il}}$ and a start time $st_{\theta_{il}}$ to be determined. Each machine can work at different rates, so the combination of processing time and energy consumption is presented by a tuple $\{p_{\theta_{il}}, e_{\theta_{il}}\}$.

A feasible schedule is composed of a complete assignment of starting times of tasks that satisfy the following constraints:

1. The tasks of each job are sequentially scheduled.
2. Each machine can process at most one task at any time.
3. No preemption is allowed.

The aim of the JSMS problems is to find a feasible schedule that minimizes makespan and energy consumption meanwhile maximizes the robustness of the schedule.

This problem represents an extension of the standard job-shop scheduling problem ($J||C_{max}$) Blazewicz et al. (1986). An association between processing time and energy has been created so the problem JSMS can be denoted as $J(Speed)||C_{max}, Energy$. For each task, three different speeds/modes (called {1,2,3}) have been defined. Each possible processing speed/mode of a machine is associated to a processing time and an energy consumption. As we have pointed out before, it is assumed that as the working speed of a machine increases, the energy consumption also increases despite the shorter processing time (Zhang and Chiong (2016)). However there is neither normalized proportional processing speed nor a direct relationship among these parameters. It is typical to have energy consumption as an exponential function of speed (Fang et al. (2013); Bouzid (2005)). However, the user could select another relationship among these parameters.

According to the classification proposed in Gahm et al. (2016), our problem can be categorized as:

- Energetic coverage: Directly reduce AES demand (PS).

- Energy demand: Job related (JR), Machine related (MR), Flexible (FLX).
- Objective criteria: Non-monetary (makespan)
- System of objectives: Multi-objective.
- Manufacturing model: Jobshop/projects cheduling (J/PS).
- Solution method: Heuristic.

4. Rescheduling by Match-up in JSMS

Once a schedule suffers an incidence and it cannot be absorbed by robustness, rescheduling is required to minimize the needed changes in the original schedule. Our main objective is to develop a technique to search for a time point called *match-up point* where the original schedule can be re-established/recovered. Thus, the rescheduling is only necessary in the period between the disruption point and the match-up point (Figure 1). It reduces the computational cost and the time needed to re-establish the schedule. At the same time, part of the schedule remains unaltered so the stability is increased and nervousness decreased. Without loss of generality, we assume that a disruption is only generated in a single machine during a task processing.

4.1. A Match-up technique

In JSMS problems, the machines can work at different speeds with their corresponding energy consumptions and processing times. This variability can be used to minimize the match-up point for each machine. Due to the fact that an incidence appears in a random machine during task execution, these incidences can be propagated and they can affect other tasks executed in other machines. The main goal of the proposed algorithm is to analyze the propagation of a disrupted task, and to accelerate each involved machine to absorb the incidence or to reduce it. Once the incidence has been absorbed, the original schedule is recovered in a given time point (match-up point). It must be taken into account

that the algorithm returns a match-up point and also a valid reschedule. However, it can be observed that the involved machines in the rescheduling process have been accelerated, so the obtained reschedule is not an energy efficient solution. Thus, a memetic algorithm will use the match-up point to minimize the energy consumption in this rescheduling zone.

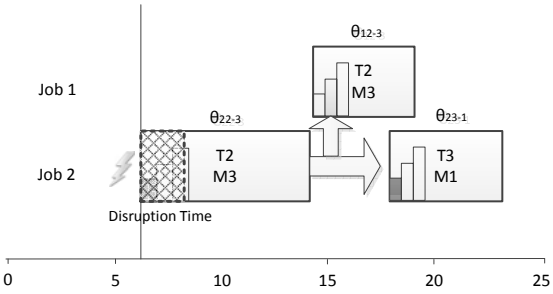

Figure 2: Task relationship by job or machine

Following the example of Figure 1, Figure 2 shows the relationship between a disrupted task θ_{22-3} and the next two related tasks. One task θ_{23-1} related with task θ_{22-3} by the precedence constraint in the same job 2, and task θ_{12-3} related with task θ_{22-3} by the same machine 3. Algorithm 1 shows the pseudocode of the match-up technique. The input of the algorithm is the original schedule (Schedule) and the incidence that occurs in a machine during task execution (IncidentTask). The output of the algorithm is the match-up point of the schedule (MatchUpSchedule) and a valid reschedule (Solution) if the incidence is absorbed, or the algorithm returns that the makespan of the original solution has been reached.

The algorithm 1 works as follow. There is a queue called InvolvedTasks that stores all tasks affected by the incidence. These tasks, inserted in the queue, store its own delay-propagated by the incidence. The first task inserted in InvolvedTasks is IncidentTask and this task stores as delay-propagated the initial disruption time (see Figure 1). Then, it is checked if next task by machine and the next task by precedence constraint are able to absorb the incidence (see

Algorithm 1: CalculateMatch-Up(Input (Schedule,IncidentTask), Output ((MatchUpSchedule,Solution)∨(MakespanReached))

```
InvolvedTasks= φ; //The queue is initialized empty
InvolvedTasks ← IncidentTask;
MakespanReached= False;
while (InvolvedTasks≠ φ and !MakespanReached) do
   CurrentTask=InvolvedTasks.pop; //Access next element in the queue
   InvolvedTasks ← InvolvedTasks\CurrentTask;
   NextTaskMach = GetNextbyMach(CurrentTask); //Select next task by machine
   NextTaskJob = GetNextbyJob(CurrentTask); //Select next task by precedence constraint
   Check(NextTaskMach, &MakespanReached);
   Check(NextTaskJob, &MakespanReached);
end while
if (!MakespanReached) then
   MatchUpSchedule = Max(MatchUp[NMach]);
   Return (MatchUpSchedule,Solution);
else
   Return MakespanReached
end if
```

Algorithm 2). If both tasks absorb the incidence, then the algorithm returns the match-up schedule, that is, the latest end time of the affected tasks. However if any task cannot completely absorb the incidence, it will be inserted in the InvolvedTask queue to be checked in next iterations. The amount of time that this task was not able to absorb is calculated and stored. Once a task is able to absorb the propagated incidence, a match-up for the involved machine is store (MatchUp[NMach]). The algorithm is iteratively executed until there is no task in the queue or the makespan of the original schedule is reached (MakespanReached= True). If the InvolvedTasks queue is empty, then the incidence has been absorbed, a match-up point of the schedule and a valid reschedule is obtained. Otherwise, the incidence has not been absorbed in the given makespan.

Algorithm 2 checks if a task is able to absorb its delay-propagated. To this end, the function *AbsorbIncidence* is committed to set the involve machine at highest speed to minimize the duration of this task, which it is updated. Thus,

Algorithm 2: Check(Input (Task,MakespanReached), Output (MakespanReached))

```
RecovTime=AbsorbIncidence(Task);
Update(end_time(task));
if (RecovTime ≥ delay-propagated(Task)) then
   UpdateMatchUp(machine(Task),end_time(Task));
else
   Delay_propagated(Task)=Delay_propagated(task)-RecovTime;
   InvolvedTasks ← Task; //The task is added in the queue.
   if (MakespanReached(Task)) then
      MakespanReached=True;
   end if
end if
Return MakespanReached;
```

the recovered time (RecovTime) is calculated. If this time is enough to recover the incidence, then the match-up point of the involve machine is updated with the new end time of this task. If not, the delay-propagated of this task is updated and the task is inserted in the queue. Finally, it is checked that the updated task has not reached the makespan of the original solution.

4.2. Rescheduling to minimize energy consumption

Once a match-up point is obtained, a new scheduling sub-problem can be defined from the incidence point until the match-up point. In the previous section a valid solution was obtained, so our aim in this section is to improve this solution in terms of energy consumption.

In Algorithm 3 the rescheduling algorithm is presented. The aim of this algorithm is to minimize the energy consumption and a makespan lower than a given threshold (the match-up point). Thus, the algorithm starts by minimizing only the energy consumption ($\lambda = 0$, see 1). If no solution is found with lower makespan that the given threshold (match-up point), then the lambda value is increased and the process is repeated. Thus, the first solution found by algorithm 3 will be the best solution found with the minimum energy consumption and makespan lower that the given threshold. If algorithm 3 finds a better so-

lution, in terms of energy consumption than the one obtained by algorithm 1, then this solution will be returned as final schedule for the given subproblem. If not, the algorithm returns the same solution obtained by algorithm 1.

Algorithm 3: Rescheduling (Sub-problem, Schedule, MatchUp

```
lambda = 0;
Makespan=MatchUp + 1;
while (lambda <=1 and Makespan>MatchUp) do
   ScheduleImp=MemeticAlgorithm (Sub-problem, λ)
   Makespan=ScheduleImp.Mk;
   lambda = lambda + 0.1;
end while
if ((ScheduleImp.Mk<=Schedule.Mk) and (ScheduleImp.En<Schedule.En)) then
   Return ScheduleImp;
else
   Return Schedule;
end if
```

4.2.1. Memetic Algorithm (GA+LS)*

In this section we present the memetic algorithm which combines a genetic algorithm (GA) with a local search (LS). Genetic Algorithms are adaptive methods which may be used to solve optimization problems Beasley et al. (1993). The pseudo code for the proposed genetic algorithm is shown in algorithm 4.

Chromosome encoding and decoding. Each candidate represents a solution in the solution space. The first step to construct the GA is to define an appropriate genetic representation (coding). In Varela et al. (2005), it is proposed a coding where a chromosome is a permutation of the set of tasks that represents a tentative ordering to schedule them, each one being represented by its job number. This encoding has a number of interesting properties for the classic job-shop scheduling problem. However, in the JSMS problem, the machine speed of each operation has to be represented. Thus, we add a value to each task in order to represent the speed of the machine that processes this task. When the chromosome representation is decoded, each task starts as soon as possible following

Algorithm 4: MemeticAlgorithm (JSMS, λ)

```
Initial-Population(Population, Size);
Evaluate-Fitness(Population);
while (Stopping criterion is not fulfilled) do
    RandomShuffle(Population);
    for (i=0; i<populationsize; i=i+2) do
        Crossover(Population[i],Population[i+1],Brother,Sister);
        Mutation(Brother,Sister,Brother',Sister');
        if (Runtime>80%timeOut) then
            LocalSearch(Brother');
            LocalSearch(Sister');
        end if
        Evaluate-Fitness(Brother',Sister');
        SaveTempPopulation(TempPopulation,Brother',Sister');
    end for
    Update-Population(Population,TempPopulation);
end while
Return Best Schedule;
```

the precedence and machine constraints. With the machine speed representation, the processing time for each task and the energy consumption can be calculated.

Initial population and Fitness. Each gene represents one task of the problem. The position of each task determines its dispatch order in this genome/solution. The initial chromosomes are obtained following some dispatching rules or by random permutation. We employ common dispatching rules such as SPT (Shortest Processing Time), LPT (longest Processing Time), JML (Job with More Load), JMT (Job with More Tasks), MML (Machine with More Load) and MMT (Machine with More Tasks). We also employ a random rule, which randomly select a job from the remaining jobs. To create each genome, each dispatching rule is randomly selected. Thus, it obtains variety and diversity in the initial population. Machine speed for each gene is generated depending on the λ value. For λ values lower than 0.6 the machine speed value is set to 1; if $\lambda = 0.6$ the speed value is set to 2; for λ values equals to 0.7, 0.8 and 0.9,

the machine speed value is set to a random value in {2,3}; and for $\lambda = 1$ the speed value is set to 3.

The definition of fitness function is just the objective function value proposed in 1. The objective is to find a solution that minimizes the multi-objective makespan and energy consumption. So the fitness function is defined as (equation 1), where the weights assigned to both variables are given by the λ value. Since the values of energy consumption and makespan are not proportional, it is necessary to normalize both measures. Makespan is divided by MaxMakespan which is the maximum makespan value in a GA execution when λ is equal to 0. MaxEnergy is the sum of the energy needed to execute all tasks at top speed.

$$F = \lambda * \frac{Makespan}{MaxMakespan} + (1 - \lambda) * \frac{SumEnergy}{MaxEnergy} \qquad \lambda \in [0, 1] \qquad (1)$$

Crossover operator. For chromosome mating, the GA uses the Job-based Order Crossover (JOX) described in Bierwirth (1995). Given two parents, JOX selects a random subset of jobs and copies their genes to the offspring in the same positions as they are in the first parent, then the remaining genes are taken from the second parent so as they maintain their relative ordering.

The remaining elements of GA are rather conventional. To create a new generation, all chromosomes from the current one are organized into couples which are mated two offsprings in accordance with the crossover probability.

Mutation operator. The two offsprings generated with the crossover operation can also be mutated in accordance the mutation probability. Two positions of chromosome child are randomly chosen (position "a" and position "b"), where "a" must be lower than "b". Values between "a" and "b" are shuffled randomly. Each position represents a task, so the tasks are shuffled randomly and also the machine speed is changed for each task be-

tween the possible speeds. Finally, tournament replacement among every couple of parents and their offsprings is done to obtain the next generation.

4.2.2. Local Search

Conventional GAs can produce good results. However, significant improvements can be obtained by hybridization with other methods. A local search algorithm starts from a solution and then iteratively moves to neighbour solutions. This is possible only if a neighbourhood relation is defined on the search space. Local search (LS) is implemented by defining a neighbourhood of each point in the search space as the set of chromosomes reachable by a given transformation rule. Then, a chromosome is replaced by the selected neighbour that satisfies the acceptance criterion. We propose a neighbourhood structure based on the concepts of critical path and critical block Matsuo et al. (1989), Van Laarhoven et al. (1992) and Nowicki and Smutnicki (1996). A critical block is a maximal subsequence of operations of a critical path requiring the same machine. In Mattfeld (1995) is defined the neighbourhood structure N_1 for JSP. It considers a set of moves called "interchange near the borderline of blocks on a single critical path", what means swapping pairs of operations only at the beginning or at the end of a critical block Mattfeld (1995).

Following the idea of neighbourhood structure N_1 and the concepts of critical path and critical block, we have defined energy-efficiency neighbourhood structure (N_{EE}) where each task in the critical path is analysed to check if the next tasks of the same machine are consecutive and involved in the critical path. If this condition is met a new neighbour is created swapping both tasks and the machine speed is increased if its fitness is not worsened. Furthermore, when a task is not in a critical path, its speed is decreased to create a new neighbour in order to save energy. LS is only carried out if the runtime is bigger than the 80% of the assigned time-out.

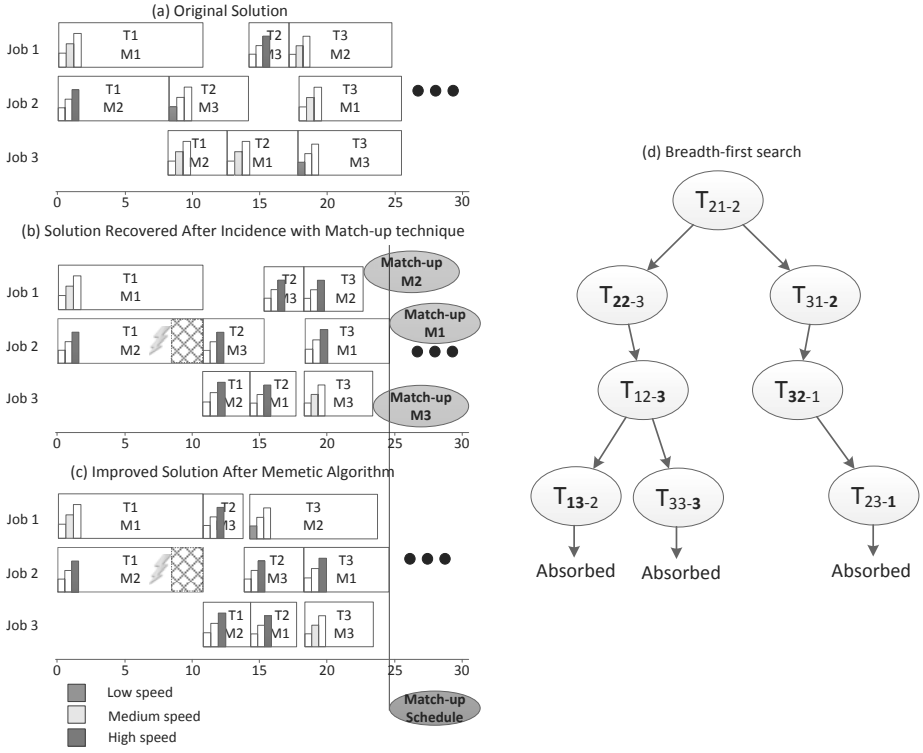

Figure 3: Original solution, recovered solution, improved solution after a disruption and breadth-first search

4.2.3. Example

Figure 3 shows an example of how algorithms work. It shows a scheduling problem with 3 jobs, 3 machines and 3 tasks per job. Figure 3a shows an optimize schedule in terms of makespan and energy consumption. The tasks in green were executed at low speed (1) so they required a large processing time; the tasks in yellow were executed at medium speed (2), so they required a medium processing time; and the tasks in red were executed at high speed (3), so they required a low processing time. Thus, given a disruption, the match-up algorithm searches for a solution by increasing the speed of some machines and using the existing buffers/gaps (see figure 3b). The match-up point of each machine is calculated and therefore the match-up point of the scheduling is obtained. Furthermore the original schedule was recovered with a valid solution. To this end, some of the involved tasks were executed at higher speed in order to reduce this match-up point. Finally, the memetic algorithm carries out improvements between the disruption point and the obtained match-up point (Figure 3c). It can be observed that the task 2 of job 1 was executed before task 2 of job 2 (both using machine 3), so task 3 of job 1 was executed at lower speed and therefore its processing time was increased and the energy consumption reduced. However these changes did not modify the rest of the schedule. Figure 3d shows the tree search carried out by the match-up technique.

5. Evaluation

In this section, an evaluation of the proposed techniques is carried out. First of all, some incidences were simulated over the schedules and robustness was measured. When the machines that suffer the incidences cannot absorb them (by robustness), our rescheduling methods are applied. These incidences are analyzed with our match-up technique to determine the temporal interval of the schedule that had to be rescheduled (sub-problem). Thus, this sub-problem is solved with the proposed memetic algorithm to obtain the best energy-efficiency schedule in a given timeout (100 seconds).

In this evaluation, the Lawrence benchmarks from OR-Library were used. Lawrence instances have 10, 15, 20 and 30 jobs (J). The number of tasks per job (V_{max}) is 5, 10 and 15, and the number of machines is equal to the number of tasks per job. The range of processing times (pi) is a value between [1, 100]. In all cases, 5 instances were considered from each combination (JxV_{max}) 10x5, 15x5, 20x5, 10x10, 15x10, 20x10, 30x10 and 15x15, with a total of 40 instances. These instances were extended (as explained in Escamilla et al. (2014)) assigning to each task three different modes/speeds, so each task can be executed in three different processing times with their corresponding energy consumptions. In all instances, the objective is the fitness function proposed in formula (1). It generates the Pareto Front by increasing the λ values. For $\lambda = 0$, the objective is to minimize energy consumption, meanwhile for $\lambda = 1$ the objective is to minimize makespan. These instances and further information about the extended benchmarks can be found in the webpage[1].

5.1. Incidences and Robustness

Once an initial solution was obtained for each instance, some incidences were simulated to measure the robustness. Thus, for each solution, 100 incidences were generated (500 for each group of instances). Once an incidence was assigned to a machine during a task execution, the duration of this disrupted task was randomly increased between 1 and 30% of the maximum processing time of this instance.

Figure 4 shows that all instances maintained a similar behaviour against the incidences. When the λ values are low the objective is mainly focused on minimizing energy, so makespan is higher and therefore, there exists many buffers/gaps between consecutive tasks. In this case, many incidences can be absorbed without rescheduling (robust schedules). It must be taken into account that for instances with less tasks (10x5, 15x5, 20x5), the robustness is lower (mainly is high λ values) because there is less variability in the allocation of

[1] http://gps.webs.upv.es/jobshop/

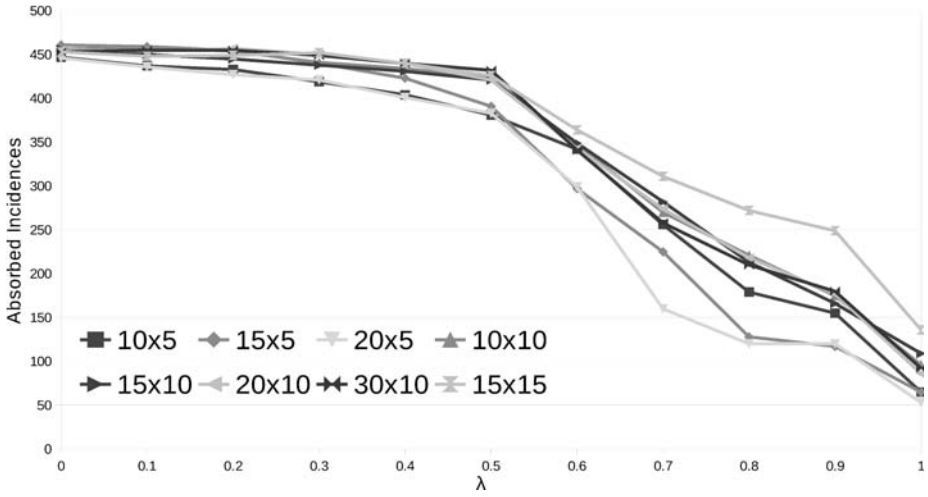

Figure 4: Absorbed incidences for different instances

tasks, so the number of buffers/gaps is lower.

5.2. Evaluating the Match-up technique

All the incidences/disruptions that were not absorbed by the own robustness of the schedule, must be managed in the rescheduling process. To this end, our match-up technique was applied to recover as soon as possible the original schedule. To this end, given an incidence, the algorithm searches for the best match-up point or returns that the makespan of the original schedule is achieved. This last case occurs when the incidence is located at the end of the schedule, so there is not enough time to recover the solution. Figure 5 shows the runtime of the Match-up technique for different group of instances. It represents the average runtime (milliseconds) to solve each type of instances over an Intel Core 2 Duo Processor with 1Gb Ram Memory.

Tables 1 and 2 show, for each group of instances, the amount of incidences (from a total of 500) that were absorbed by robustness (Rb), by our match-up techniques (MUp) or not recovered because the makespan was reached (Mk). It can be observed the importance of the λ value. For low λ values, the objective is to minimize energy consumption so most machines work at low speed

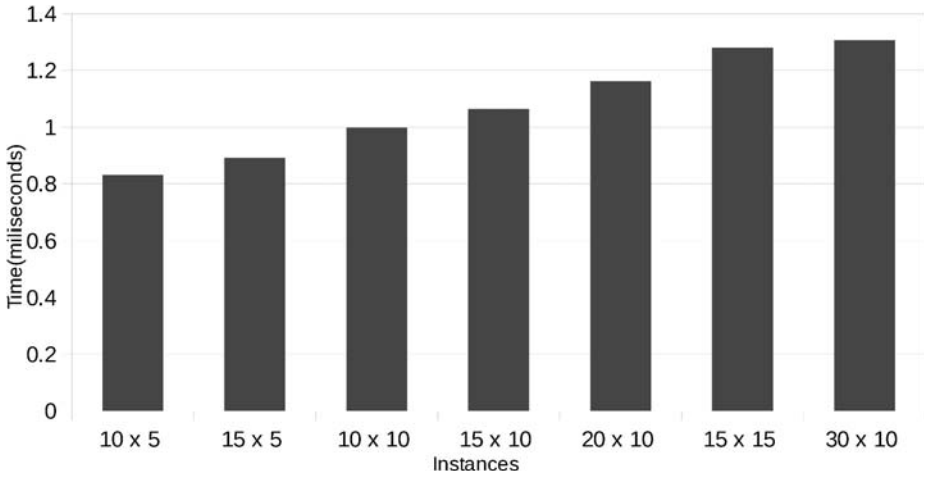

Figure 5: Average runtime of Match-up technique for different instances

Table 1: Number of incidences absorbed by robustness (Rb) by the match-up point (MUp) or not recovered (Mk) for different instances

	10x5			15x5			20x5			10x10		
λ	Rb	MUp	Mk	Rb	MUp	Mk	Rb	MUp	Mk	Rb	MUp	Mk
0	447	40	13	461	35	4	446	53	1	456	27	17
0.1	437	55	8	459	40	1	436	61	3	458	24	18
0.2	433	56	11	455	45	0	427	66	7	454	33	13
0.3	419	67	14	440	56	4	421	72	7	441	42	17
0.4	404	70	26	423	71	6	401	95	4	434	48	18
0.5	381	95	24	391	100	9	383	112	5	423	57	20
0.6	342	113	45	297	170	33	299	169	32	348	97	55
0.7	256	165	79	225	217	58	160	252	88	270	128	102
0.8	179	170	151	128	206	166	120	204	176	221	120	159
0.9	155	137	208	117	209	174	120	199	181	175	83	242
1	65	141	294	65	235	200	53	177	270	96	109	295

Table 2: Number of incidences absorbed by robustness (Rb) by the match-up point (MUp) or not recovered (Mk) for different instances

	15x10			20x10			30x10			15x15		
λ	Rb	MUp	Mk	Rb	MUp	Mk	Rb	MUp	Mk	Rb	MUp	Mk
0	456	38	6	455	38	7	453	45	2	453	28	19
0.1	450	42	8	454	44	2	455	45	0	448	27	25
0.2	445	46	9	457	40	3	455	44	1	449	29	22
0.3	438	54	8	450	48	2	449	50	1	452	29	19
0.4	431	64	5	440	49	11	440	55	5	440	43	17
0.5	421	63	16	421	79	0	432	63	5	426	49	25
0.6	349	114	37	343	140	17	341	146	13	364	78	58
0.7	282	177	41	274	193	33	258	214	28	311	119	70
0.8	213	222	65	218	230	52	210	253	37	272	145	83
0.9	166	196	138	177	237	86	180	271	49	249	128	123
1	109	171	220	86	180	234	92	234	174	136	138	226

and the makespan increase. This makes that many buffers/gaps appear along the schedule and most incidences are absorbed by using these buffers (robust solutions), so no rescheduling is needed. It can be observed in Tables 1 and 2 that as the λ values increased, the number of incidences absorbed by robustness (Rb) decreased meanwhile the number of incidences absorbed by our match-up techniques (MUp) increased. The highest values of MUp were reached for λ values of 0.7, 0.8 or 0.9, when the objective is to mainly focused on minimizing makespan and there are not many buffers/gaps in the original schedule to absorb the incidences by robustness, so our match-up techniques could recover more schedules. It must be taken into account that for $\lambda = 1$, the objective is only focused on minimizing makespan, so all machines are working at highest speed and it is less probably to recover the solution because the makespan is achieved.

5.3. Evaluating the Memetic algorithm for rescheduling

In this evaluation, we consider that the match-up technique has achieved a match-up point and therefore an initial recovered schedule has been obtained. Thus, the objective of the memetic algorithm is to improve the solution obtained in terms of energy efficiency in a given timeout fixed to 100 seconds.

Table 3: Rescheduling in Match-up results

	15x10					30x10				
	MUp	Rec	%Rec	EnRed	%EnRed	MUp	Rec	%Rec	EnRed	%EnRed
0	38	27	71.1%	3960	35.62%	45	30	66.7%	5318	43.82%
0.1	42	34	81.0%	6512	39.05%	45	38	84.4%	5756	46.12%
0.2	46	31	67.4%	6050	41.74%	44	31	70.5%	7061	47.14%
0.3	54	36	66.7%	6672	38.89%	50	32	64.0%	6308	43.35%
0.4	64	52	81.3%	11599	43.48%	55	40	72.7%	7549	44.95%
0.5	63	49	77.8%	9472	39.27%	63	46	73.0%	10281	44.07%
0.6	114	94	82.5%	18413	30.11%	146	105	71.9%	18374	22.36%
0.7	177	119	67.2%	22903	19.92%	214	103	48.1%	16272	7.23%
0.8	222	102	45.9%	21782	7.78%	253	86	34.0%	15260	4.93%
0.9	196	64	32.7%	87961	25.36%	271	55	20.3%	24391	5.04%
1	171	114	66.7%	187027	74.52%	234	177	75.6%	739241	86.08%

Table 3 shows the results for instances 15x10 and 30x10 which are considered representative instances. The column (MUp) represents the total number of instances recovered by the match-up technique, the columns (Rec) and (%Rec) represent the number and the percentage of recovered instances that the memetic algorithm was able to reduce the energy consumption, respectively. Finally, the columns Energy-Reduced and %Energy-reduced represent the amount of energy and the percentage of energy that the memetic algorithm was able to reduce in the obtained schedule with respect to the schedule recovered by the match-up technique.

It can be observed in Table 3 that the memetic algorithm was able to reduce the energy consumption in a significative number of incidences. For low λ values, the number of instances that the memetic algorithm was able to reduce the energy consumption did not vary significatively (around 40). However, when λ is equal to 0.6 and 0.7, the values of (Rec) increased and when λ is

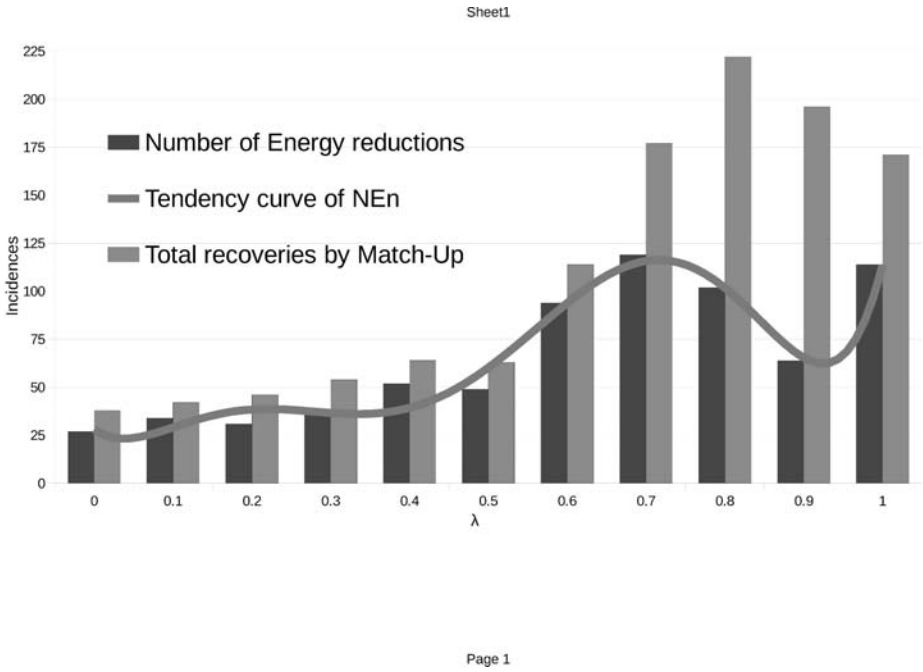

Figure 6: Comparative of Mu and NEn for instance 10x5

equal to 0.8 and 0.9, the values of (Rec) decreased. This behavior is justified by the characteristic of the original schedule. For high λ values, the (MUp) values were high because robustness was low, so there were more options for rescheduling. Thus, more rescheduling processes were carried out for λ equals to 0.6 and 0.7 and therefore the memetic algorithm was able to reduce the energy consumption in a high percentage of instances. However, when λ is equal to 0.8 and 0.9 this percentage decreased, because the fitness function is mainly focused on minimizing makespan. Thus, most machines are working at highest speed so energy used cannot be reduced. Finally, when $\lambda = 1$, it is considered an special case because the makespan in only taken into consideration in the objective function, so many tasks that are not involved in the critical path can be executed at lower speed without worsening the makespan. This behavior can be highlighted in Figure 6 with the tendency curve of (Rec).

Table 3 also shows the total energy saved during the energy reduction carried

out by the memetic algorithm. The values of Energy Reduced (EnRed) increased when λ increased because for high values of λ, the energy used was higher so it was possible to reduce more energy. The column (%EnRed) represents the percentage of energy saved. It can be observed that for low λ values, the percentage of energy reduced did not vary significatively. However, when λ is equal to 0.6, 0.7 and 0.8, the percentage decreased and when λ is equal to 0.9, the percentage increased again. This behavior is also related with the characteristic of the original schedule because for λ equals to 0.6 and 0.7 the memetic algorithm was able to reduce the energy consumption in a high percentage of instances but not a high quantity. For these λ values, the energy used was not too high but for $\lambda = 0.9$, the technique was able to save more amount of energy.

6. Conclusion

Manufacturing industries involve a large number of scheduling problems. Most of these problems are dynamic so they face with incidences so, recovery techniques are needed to re-establish the original scheduling as soon as possible. Moreover, industries are facing increasing requirements of sustainability, so energy consumption processes should be minimized.

In this paper, we propose two different techniques to manage rescheduling over an extended version of the job-shop scheduling problem. Thus, given an incidence, a first technique, called match-up technique, is committed to determine the time point of the schedule where the original solution is recovered and a non-energy efficient solution is obtained. Afterwards, a memetic algorithm is proposed to search for an energy efficient solution in the established rescheduling zone. An extensive study was carried out to analyze the behavior of the proposed techniques. To this end, some incidences were simulated over some well-known benchmarks. The proposed match-up technique maintained a good performance and many instances were recovered in an efficient way. Finally most of the rescheduling solutions were improved to save more energy consumption. It can be seen that upon a disruption, different rescheduling solutions can be

obtained , all of them holding with the requirement of the initial makespan, but with different values of energy consumption. These techniques can be applied in real industry where minor disruptions daily occurs and the original schedule must be reestablished as soon as possible to reduce nervousness and improve in stability, as well as energy consumption and sustainability

Acknowledgment

This research has been supported by the Seventh Framework Programme under the research project TETRACOM-GA609491 and the Spanish Government under research project TIN2013-46511-C2-1.

References

References

Abumaizar, R.J., Svestka, J.A., 1997. Rescheduling job shops under random disruptions. International Journal of Production Research 35, 2065–2082.

Akturk, M.S., Gorgulu, E., 1999. Match-up scheduling under a machine breakdown. European journal of operational research 112, 81–97.

Arnaout, J.P., 2014. Rescheduling of parallel machines with stochastic processing and setup times. Journal of Manufacturing Systems 33, 376–384.

Barber, F., Salido, M.A., 2015. Robustness, stability, recoverability, and reliability in constraint satisfaction problems. Knowledge and Information Systems 44, 719–734.

Beasley, D., Martin, R., Bull, D., 1993. An overview of genetic algorithms: Part 1. fundamentals. University computing 15, 58–58.

Bierwirth, C., 1995. A generalized permutation approach to jobshop scheduling with genetic algorithms. OR Spectrum 17, 87–92.

Blazewicz, J., Cellary, W., Slowinski, R., Weglarz, J., 1986. Scheduling under resource constraints-deterministic models. Annals of Operations Research 7, 1–356.

Bouzid, W., 2005. Cutting parameter optimization to minimize production time in high speed turning. Journal of Materials Processing Technology 161, 388–395.

Church, L.K., Uzsoy, R., 1992. Analysis of periodic and event-driven rescheduling policies in dynamic shops. International Journal of Computer Integrated Manufacturing 5, 153–163.

Draganescu, F., Gheorghe, M., Doicin, C., 2003. Models of machine tool efficiency and specific consumed energy. Journal of Materials Processing Technology 141, 9–15.

Escamilla, J., Salido, M.A., Giret, A., Barber, F., 2014. A metaheuristic technique for energy-efficiency in job-shop scheduling. Workshop on Constraint Satisfaction Techniques (COPLAS) 2014: 24th International Conference on Automated Planning and Scheduling (ICAPS) , 42–50.

Fang, K., Uhan, N., Zhao, F., Sutherland, J., 2013. Flow shop scheduling with peak power consumption constraints. Annals of Operational Research 206, 115–145.

Gahm, C., Denz, F., Dirr, M., Tuma, A., 2016. Energy-efficient scheduling in manufacturing companies: A review and research framework. European Journal of Operational Research 248, 744–757.

Hall, N.G., Potts, C.N., 2004. Rescheduling for new orders. Operations Research 52, 440–453.

Herroelen, W., Leus, R., 2004. Robust and reactive project scheduling: a review and classification of procedures. International Journal of Production Research 42, 1599–1620.

Li, R.K., Shyu, Y.T., Adiga, S., 1993. A heuristic rescheduling algorithm for computer-based production scheduling systems. The International Journal Of Production Research 31, 1815–1826.

Liu, Y., Dong, H., Lohse, N., Petrovic, S., Gindy, N., 2014. An investigation into minimising total energy consumption and total weighted tardiness in job shops. Journal of Cleaner Production 65, 87–96.

Matsuo, H., Suh, C.J., Sullivan, R.S., 1989. A controlled search simulated annealing method for the single machine weighted tardiness problem. Annals of Operations Research 21, 85–108.

Mattfeld, D.C., 1995. Evolutionary Search and the Job Shop Investigations on Genetic Algorithms for Production Scheduling. Springer-Verlag.

May, G., Stahl, B., Taisch, M., Prabhu, V., 2015. Multi-objective genetic algorithm for energy-efficient job shop scheduling. International Journal of Production Research 53, 7071–7089.

Mestl, H.E., Aunan, K., Fang, J., Seip, H.M., Skjelvik, J.M., Vennemo, H., 2005. Cleaner production as climate investmentintegrated assessment in taiyuan city, china. Journal of Cleaner Production 13, 57–70.

Mouzon, G., Yildirim, M., Twomey, J., 2007. Operational methods for minimization of energy consumption of manufacturing equipment. International Journal of Production Research 45, 4247–4271.

Neugebauer, R., Wabner, M., Rentzsch, H., Ihlenfeldt, S., 2011. Structure principles of energy efficient machine tools. CIRP Journal of Manufacturing Science and Technology 4, 136–147.

Nowicki, E., Smutnicki, C., 1996. A fast taboo search algorithm for the job shop scheduling problem. Management Science 42, 797–813.

Pujawan, I.N., 2004. Schedule nervousness in a manufacturing system: a case study. Production planning & control 15, 515–524.

Qi, X., Bard, J.F., Yu, G., 2006. Disruption management for machine scheduling: the case of spt schedules. International Journal of Production Economics 103, 166–184.

Salido, M.A., Escamilla, J., Barber, F., Giret, A., Tang, D., Dai, M., 2013. Energy-aware parameters in job-shop scheduling problems. GREEN-COPLAS 2013: IJCAI 2013 Workshop on Constraint Reasoning, Planning and Scheduling Problems for a Sustainable Future , 44–53.

Steele, D.C., 1975. The nervous mrp system: how to do battle. Production and Inventory Management 16, 83–89.

Subramaniam, V., Raheja, A.S., 2003. maor: A heuristic-based reactive repair mechanism for job shop schedules. The International Journal of Advanced Manufacturing Technology 22, 669–680.

Tonelli, F., Bruzzone, A., Paolucci, M., Carpanzano, E., Nicol, G., Giret, A., Salido, M., Trentesaux, D., 2016. Assessment of mathematical programming and agent-based modelling for off-line scheduling: application to energy aware manufacturing. CIRP Annals Manufacturing Technology , to appear.

Van Laarhoven, P.J., Aarts, E.H., Lenstra, J.K., 1992. Job shop scheduling by simulated annealing. Operations research 40, 113–125.

Varela, R., Serrano, D., Sierra, M., 2005. New codification schemas for scheduling with genetic algorithms, in: Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach. Springer, pp. 11–20.

Vieira, G.E., Herrmann, J.W., Lin, E., 2000. Predicting the performance of rescheduling strategies for parallel machine systems. Journal of Manufacturing Systems 19, 256–266.

Vieira, G.E., Herrmann, J.W., Lin, E., 2003. Rescheduling manufacturing systems: a framework of strategies, policies, and methods. Journal of scheduling 6, 39–62.

Wu, H.H., Li, R.K., 1995. A new rescheduling method for computer based scheduling systems. International journal of production research 33, 2097–2110.

Wu, S.D., Storer, R.H., Pei-Chann, C., 1993. One-machine rescheduling heuristics with efficiency and stability as criteria. Computers & Operations Research 20, 1–14.

Yusoff, S., 2006. Renewable energy from palm oil–innovation on effective utilization of waste. Journal of cleaner production 14, 87–93.

Zhang, R., Chiong, R., 2016. Solving the energy-efficient job shop scheduling problem: a multiobjective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption. Journal of Cleaner Production 112, 3361–3375.