Integrated Intelligent Method for Solving Multi-objective MPM Job Shop Scheduling Problem[☆]

D.C. Tselios[a,1], I.K. Savvas[b,2], M-T. Kechadi[a,3,*]

[a] *UCD School of Computer Science and Informatics, University College Dublin, Belfield, Dublin 4, Ireland*

[b] *Dept. of Computer Science and Engineering, TEI of Thessaly, Zip 411 10 Larissa, Greece*

Abstract

The project portfolio scheduling problem has become very popular in recent years. Current project oriented organisations have to design a plan in order to execute a set of projects sharing common resources such as personnel teams. These projects must, therefore, be handled concurrently. This problem can be seen as an extension of the job shop scheduling problem; the multi-purpose job shop scheduling problem. In this paper, we propose a hybrid approach to deal with a bi-objective optimisation problem; Makespan and Total Weighted Tardiness. The approach consists of three phases; in the first phase we utilise a Genetic Algorithm (GA) to generate a set of initial solutions, which are used as inputs to recurrent neural networks (RNNs) in the second phase. In the third phase we apply adaptive learning rate and a Tabu Search like algorithm with the view to improve the solutions returned by the RNNs. The proposed hybrid approach is evaluated on some well-known benchmarks and the experimental results are very promising.

Keywords: project scheduling, job shop scheduling, recurrent neural network, multi-objective, genetic algorithm, adaptive learning

[☆] This document is a collaborative effort.

[*] Corresponding author

Email addresses: Dimitrios.Tselios@ucdconnect.ie (D.C. Tselios), savvas@teilar.gr (I.K. Savvas), tahar.kechadi@ucd.ie (M-T. Kechadi)

[1] Dimitrios C. Tselios is PhD student at University College Dublin.

[2] Ilias K. Savvas currently is Associated Professor at TEI of Thessaly.

[3] Tahar Kechadi currently is Professor at University College Dublin.

1. Introduction

The project portfolio scheduling focuses on discrete projects (Tselios et al., 2013*b*), and it has attracted significant research interest (Browning & Yassine, 2010; Trojet et al., 2010; Schwindt, 2005; Brucker et al., 1999) because the modern organisations operate in multi-project and multi-objective environment. The typical approach to this problem usually aims at a single objective. However, recent studies (Tselios et al., 2013*b*,*a*) have proposed multi-objective models that are close to the real world projects.

Job shop scheduling problem (JSSP) has been explored by many researchers during the last few decades. Some of these efforts utilised a very promising concept, the recurrent neural network (RNN) (Kechadi et al., 2013). Other researchers (Agarwal et al., 2011) combined two evolutionary approaches i.e. Neural Networks and Genetic Algorithms in order to solve similar problem; the Resource Constraint Project Scheduling problem. According to this approach, the key difference between the two techniques is that GA is suitable for global search while neural network fits well with local search. This distinction is also strong for our problem because we need to enhance a RNN method, as it depends on the initial conditions e.g. the initial solution. Hence, we need a method that will feed the RNN with good initial solutions and the GA approach can provide a set of good initial solutions.

In this paper, not only we propose an approach that combines two well known techniques; the neural network and the Genetic Algorithm (GA) in order to deal with the multi-purpose machines JSSP (MPM JSSP), but also we propose an auxiliary tabu search like algorithm (TSA) to improve the final solution.

Tabu search is a meta-heuristic technique (Chelouah & Siarry, 2000) which has been heavily applied to various combinatorial optimisation problems for its fast and aggressive search overcoming the limitations of the neighbour search (Cavin et al., 2004). The goal of an optimisation problem is to optimise an objective that consists of more than one function (multi-objective or vector optimisation) by selecting the best from the solution space. Exact techniques for exploring the whole solution space are NP-complete, therefore, heuristic methods could provide a good alternative that can return good solutions within a reasonable execution time. The basic concept of Tabu search aims to continue the search whenever a local optimum is reached by allowing non-improving moves. Tabus prevent the examination of previously visited solutions while the non-tabu list concentrates only on promising searches.

Table 1: Notations

Symbol	*Definition*
N	Number of projects
X	Total number of the activities
w_i	Penalty cost (weight) of project P_i
a_{ix}	x activity of project P_i
ϕ_{ix}	Load of activity a_{ix}
r_{ix}	Resources that can perform the activity a_{ix}
S_{ix}	Start time of activity a_{ix} for Makespan
S_{ix}^m	Start time of activity a_{ix} assigned to resource r_m
C_{ix}	Finish (complete) time of activity a_{ix}
$R = \{r_1, r_2, \ldots, r_M\}$	Set of the resources
M	Number of the resources
C_i	Scheduling time of project P_i
C_{max}	Scheduling time of the Project Portfolio
F_o	Objective function
T_i	Tardiness of project P_i
ξ	Total number of the constraints
K_o	Penalty factor
E_o	Energy function
E_{Comp}	Energy function for all objectives
μ_o	Learning parameter of the RNNs
ρ_o	Activation function

Using a combination of RNNs with the Tabu meta-heuristic method is a very promising technique since Tabu search could accelerate the process by not allowing returning back to already visited solutions.

The motivation of the current work is its subject matter. It deals with projects that are implemented by an organisation not just a single isolated project. Secondly, it utilises an MPM JSSP model that fits well in other disciplines such as manufacturing. Another fascinating aspect of this work is the combination of GA, RNN, Tabu and it aims at solving a multi-objective formulation of the problem.

The rest of the text is organised as follows: The system model is presented in Section 2. In Section 3 we discuss the adopted and adapted benchmark instances. In Section 4 we present the proposed approach while in Section 5 we present the experimental results and corresponding analysis. Finally, in Section 6 we conclude and discuss some future research directions.

2. System Model

A globally accepted notation for theoretical study of scheduling problems was proposed by Graham et al. Graham R. L. et al. (1979). According to this classification $\alpha|\beta|\gamma$, the generalised version of our problem can be expressed as follows:

$$\alpha_1 = PMPM, \alpha_2 = k|\beta_2 = chains, \beta_5 = d_i|\gamma = \min_{o=1}^{2} Objective_o$$

The system model of the project portfolio scheduling problem is adopted from Tselios et al. (2013*b*,*a*). Briefly, each schedule S is a vector that consists of the start times of the activities (operations) of all projects (jobs). So, the generalised constrained model can be expressed as follows:

$$\min_{o=1}^{2} Objective_o$$

Subject to

$$\begin{array}{rcll} S_{ix} - S_{ix+1} + \phi_{ix} & \leq & 0, & i \leq N, x \leq X-1 \\ S_{ix} & \geq & 0, & i \leq N, 1 \leq x \leq X \\ \delta^m_{ix,jy}(S^m_{ix} - S^m_{jy} + \phi_{ix}) & \leq & 0, & \forall r_m \in r_{ix} \cap r_{jy}, i \neq j \\ (1 - \delta^m_{ix,jy})(S^m_{jy} - S^m_{ix} + \phi_{jy}) & \leq & 0, & \forall r_m \in r_{ix} \cap r_{jy}, i \neq j \\ S^m_{ix} & = & S_{ix}, & \forall m \in R \end{array}$$

where $\delta^m_{ix,jy} = 1$ if $S^m_{ix} \leq S^m_{jy}$ otherwise $\delta^m_{ix,jy} = 0$. Note that the activities a_{ix} and a_{jy} can be assigned to the same resource r_m. The studied objectives are the Makespan (C_{max}) and the Total Weighted Tardiness ($\sum_{i=1}^{N} w_i T_i$). Table 1 summarises the notations and notions used in this paper.

3. Benchmarks

We have two ways of selecting and using benchmarks. One option is the conception and construction of a new set of benchmark instances aiming at the specific category of problems. The second alternative is the exploitation of a well known benchmarks that have been used successfully in other similar problems. We opted for the second as our problem, beside its complexity and its constraints, is part of the job-shop problems. However, the selected

benchmarks need to be adapted in order to serve the two objectives of our system model and mainly due to the second objective; the Total Weighted Tardiness.

More specifically, the majority of the instances are designed for a typical JSSP. Unfortunately, the MPM JSSP, which fits well our problem does not have a globally established benchmark instances. The work of (Bernd Jurisch, 1992), which has attracted significant interest in the last two decades, extended well known benchmarks instances (Lawrence, 1994), in order to be utilised for the MPM JSSP problem. The derived set is obtained by random possible assignments of the resources (machines) to the activities. However, this set aims at only one objective; the Makespan. Very few research works (Demirkol, E. et al., 1998) were dedicated to the Tardiness objective, but to the best of our knowledge, there is not well established sets for the second objective of our problem, which is the Total Weighted Tardiness.

Hence, we had to modify the benchmark instances to serve both objectives. Our approach exploited the method used by Dermikol et al. (Demirkol, E. et al., 1998) in order to obtain instances suitable for the Total Weighted Tardiness. We used a random generator to produce for every instance a corresponding set of weights and the due times for all jobs (projects). So, the derived benchmarks consists of the following subsets (we keep the initial code names for the instances):

- la01-la20
- mt06, mt10, mt20
- car1-car8

These benchmarks have their advantages and limitations. While they give more freedom, we cannot compare our results to the previous works as they did not use the same benchmarks. However, we still can measure the performance of our approach by calculating how close the returned solutions are to the optimal ones.

4. Proposed Approach

The overall approach of the solving method aiming at the problem consists of three distinguished phases that perform in serial fashion. In other words, the outcome of each phase is the input of the next one. More specifically,

the phases are the Initial Schedule derivation, the Generation of a set of Schedules, and the core RNN learning process. In Figure 1, the three phases are presented within their input and output.

Regarding the first phase, we need to underline that RNN based search approaches require a start point, i.e. a initial solution. So, the first step of the solving process is the construction of the initial schedules. In order to complete this stage, we used a greedy technique that gives good schedules (better than the random schedule). The key issue of this technique is to assign to each activity a resource without violating the constraints of the problem. The criterion for this choice is that the shortest or the longest first unassigned activity from all projects is assigned to the first available resource that can execute it. The technique and its results are described in detail in a previous work (Tselios et al., 2013*a*).

The second phase is responsible for the enrichment of the set of the schedules that will be used as start points by the RNN learning process. This phase receives as input the two initial schedules that were produced in phase 1 and then by employing a Genetic Algorithm which generates a number of schedules, completes a set of initial feasible solutions. In other words, the GA, that is used, enhances the diversity of the initial set of solutions.

Finally, the third phase is the main phase of the overall solving method. This phase implements the RNN learning process within an additional heuristic technique, the Tabu Search, in order to improve its performance.

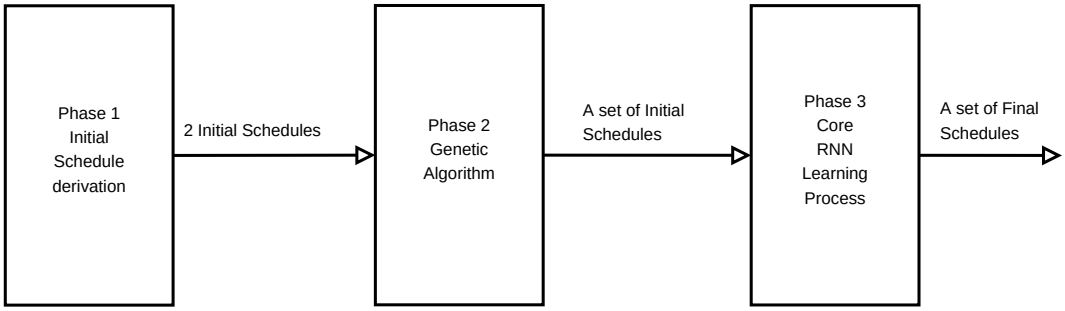

Figure 1: The overall solving method is separated into phases.

4.1. Genetic Algorithm

The Algorithm that produces initial schedules (Tselios et al., 2013*a*,*b*) returns very good start points to the RNN for many problems instances.

However, the algorithm does not perform well for large scale and complicated instances giving distant points of the solution space.

Apparently, the main cause of this effect is due to the assignment part of the algorithm and a tool which can help to amend the solution process is the employment of a Genetic Algorithm that will change the initial resource assignment deriving more and better initial schedules.

A Genetic Algorithm (GA) is an adaptive search technique that can perform in large solution spaces. According to some researchers (Agarwal et al., 2011), this technique is convenient for global searching. On the other hand, it is not so effective for searching in local level because GA produce new solutions by perturbation that leads to new distant points (solutions) of the solution space.

According to GA terminology, one significant ingredient is the solution's encoding that called chromosome. The chromosome of our problem is the schedule. Each gene of the chromosome corresponds to a specific activity of the schedule and the assigned resource as it is presented in Figure 2. Actually, each gene is a quadruple that consists of the necessary info regard to an activity. More specifically, the gene contains the project id, the phase id, the start time of the activity, and the resource identifier which has been assigned to it.

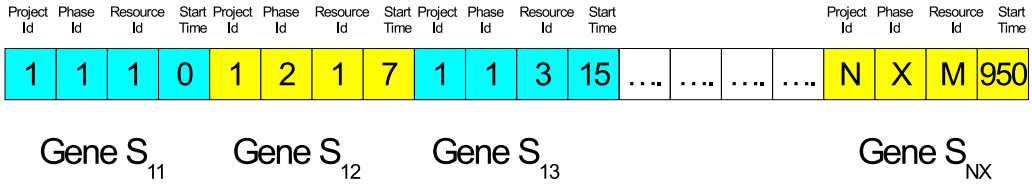

Figure 2: The form of a typical chromosome.

The operation of the GA as adapted for our scheduling problem is presented as a flow chart in Figure 3.

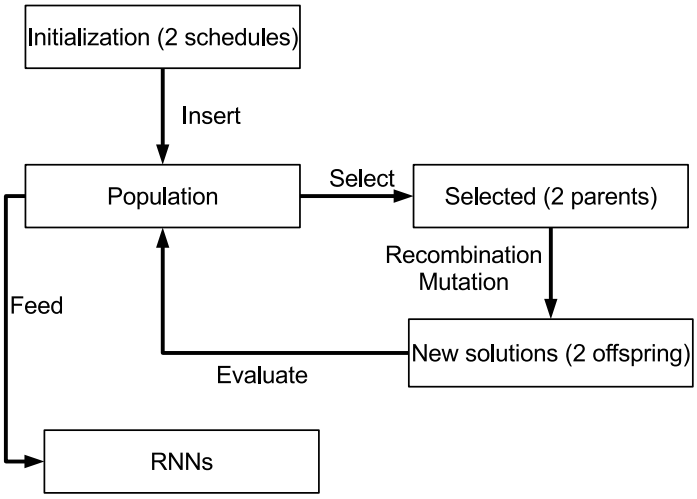

Figure 3: Flow chart diagram describing GA.

The Algorithm 1 describes, step by step, the proposed GA.

We have to clarify some of these steps, i.e. how the typical Genetic Algorithm deals with the specific scheduling problem. These steps are as follows:

1. Start: This step generates two solutions using the initial algorithm (Tselios et al., 2013*a*). We utilized this algorithm by applying two opposite rules. The first schedule derived by assigning the longest activity first and the second by assigning the shortest activity first. The schedules derived by this execution are called Initial1 and Initial2.
2. Fitness: The next step is responsible for the fitness evaluation of each schedule in the population regarding to the first objective function, the Makespan.
3. New offspring: In this step two new solutions, i.e. two schedules-offspring are created by repeating the following steps until the new schedules are successful completed.
   (a) Selection: It selects the two best schedules of the population (the criteria are their fitness value).

Algorithm 1 Genetic Algorithm

1: $POPULATION \leftarrow \{Initial1, Initial2\}$
2: EVALUATE $Initial1$, $Initial2$
3: $PopulationCounter \leftarrow 2$
4: **while** $PopulationCounter \leq MAXCHROMOSOME$ **do**
5: SELECT the best two solutions $parent1$, $parent2$
6: CROSSOVER randomly parts of $parent1$, $parent2$
7: FORM $offspring1$, $offspring2$
8: MUTATE $offspring1$, $offspring2$
9: EVALUATE $offspring1$, $offspring2$
10: **if** $(Evaluation(offspring1) < (Evaluation(parent1)$ and $(Evaluation(offspring1) < (Evaluation(parent2)$ **then**
11: $POPULATION \leftarrow POPULATION \bigcup \{offspring1\}$
12: $PopulationCounter \leftarrow PopulationCounter + 1$
13: **end if**
14: **if** $(Evaluation(offspring2) < (Evaluation(parent1)$ and $(Evaluation(offspring2) < (Evaluation(parent2)$ **then**
15: $POPULATION \leftarrow POPULATION \bigcup \{offspring2\}$
16: $PopulationCounter \leftarrow PopulationCounter + 1$
17: **end if**
18: **end while**

(b) Crossover: It forms two new schedules by copying parts of the two schedules-parents. The selection of the exchanged part is accomplished randomly.

(c) Mutation: In this process the new schedules-offspring are mutated at some points-positions in chromosome. This is the crucial part of the algorithm. Actually, this part ensures the diversity of the possible initial solutions. The mutation is applied by altering the resource assignment of a number of randomly chosen genes (activities). The exact number of the altered genes is determined by the resource density of the instance problem. In other words, this number increases when there are many potential resources that can be assigned to the activity.

(d) Accepting: If the schedules-offspring is better than its parents then the new schedules are inserted into the population.

4. Replace: The new offspring is added to the population which is updated

for the further algorithm's execution.

5. Test: If the maximum number of chromosomes is met then the algorithm is terminated otherwise it is continued from the second step.

Our choice for the solution's (schedule's) encoding is its encoding as an integer. Because the derived genes (resources assignments) after the crossover, and especially after the mutation sometimes form non-feasible schedules then we made adjustment corrections or discard the problematic offspring. Actually, we utilized the adjustment algorithm, presented on the next section, which keeps the schedules feasible. Moreover, our choice for the population size is equal to ten schedules because larger number will affect significantly the total algorithm's speed and we utilized the two-points crossover as selection method for offspring production.

The crossover, using two-points selection, is presented graphically in Figure 4.

4.2. Neural Network

The proposed RNN has only one energy function, which is the sum of the two energy functions but the objective is not a linear combination of the two objective functions. Similarly, there is only one equation of motion for both objectives, the Makespan and Total Weighted Tardiness. The altered model uses a single RNN for all objective functions. This combined form of the previous objectives is significantly different than previous approaches (Tselios et al., 2013*a*) where each objective has a dedicated RNN. Actually, the proposed RNN is an ensemble of RNN. The combined RNN can be extended to cover more than 2 objectives.

Note that this RNN does not have hidden layers and consists of only one layer of neurons with zero input. The number of the RNN neurons is calculated easily by the product $2 \times N \times X$ where N is the number of projects and X the number of activities for every project. The energy functions and the equations of motion for all RNNs, defined by the equations (1), (2):

$$E_{Comp}(S) = \sum_{o=1}^{2} F_o(S) + \sum_{o=1}^{2} K_o L_o(S) \tag{1}$$

$$\frac{dS_{ij}}{dt} = -\sum_{o=1}^{2} \mu_o \frac{\partial E_{comp}(S)}{\partial S_{ij}} \tag{2}$$

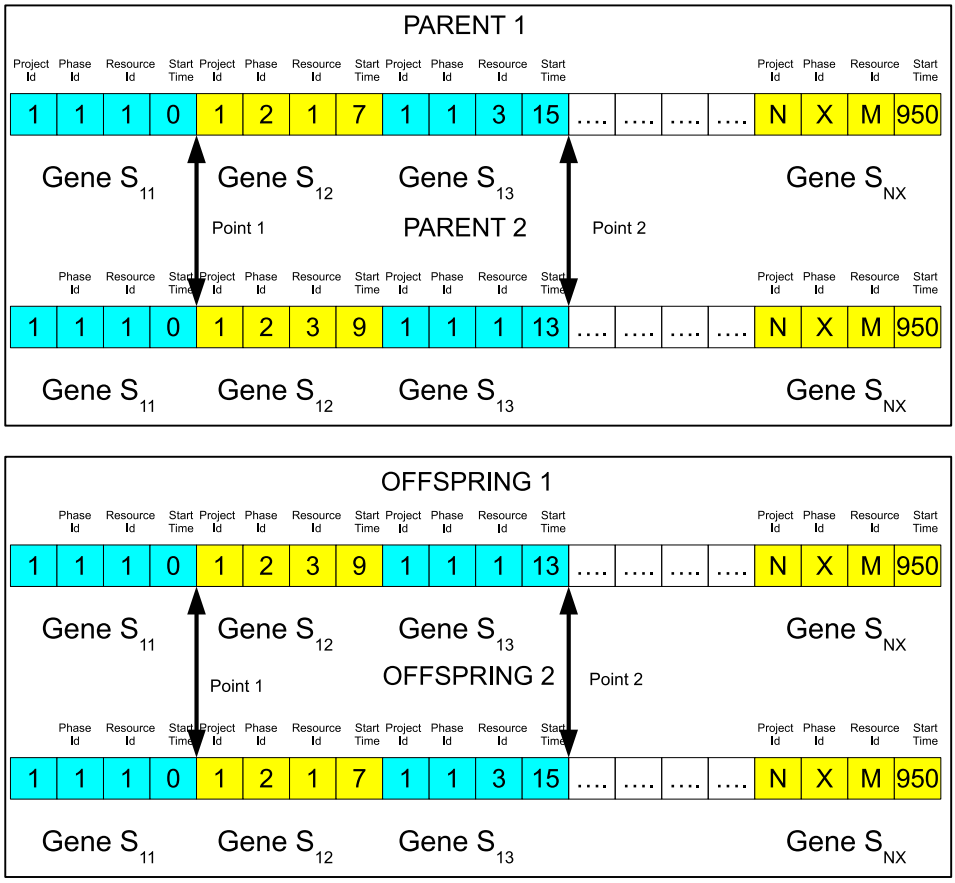

Figure 4: Two-points crossover.

The above function (1), more analytically is expressed as (3),:

$$E_{Comp}(S) = \sum_{o=1}^{2} Objective_o + \sum_{o=1}^{2} K_o \sum_{i=1}^{\xi} \rho_o[c_i(S)] \tag{3}$$

Where $c_i(S)$ are the transformations used in order to express the above problem into an unconstrained form:
$c_\gamma(S) = S_{ix} - S_{ix+1} + \phi_{ix}, c_{\gamma+2\omega}(S) = \delta(S^m_{ix} - S^m_{jy} + \phi_{ix})$
where, γ, ω, and $\xi = \gamma + 2\omega$ are respectively the number of conjunctive constraints, the number of disjunctive constraints, and the total number of constraints is (δ is equal to 0 or 1). More details about the terms can be found in table 1.

This transformation is the result of the constraints' relaxation but this type of formulation has a serious drawback. There are two competitive terms that require minimisation. Unfortunately, this antagonism provokes too many local minima. Hence, every similar approach needs a mechanism that deviates the local minima as much as possible.

Most of the existing research works were focused on single objective function problems. Adibi et al (Adibi et al., 2010) tried to minimise concurrently two different objectives; the Makespan and the tardiness of the JSSP. The interesting point of this study is that the authors combined the two objectives into one using a linear combination. We believe that this choice is not well justified and it is quite arbitrary. However, this work (Adibi et al., 2010) is one of the rare endeavors that aim at tackling JSSP using neural network technique for minimising multi-objective functions.

Although most of the machine learning systems use predefined fixed learning rates, this choice affects significantly the convergence speed. Therefore, some researchers adopted more flexible learning rates that can be changed during the neural network evolution. This approach is called dynamic learning rate (Zhang et al., 2012) or adaptive learning rate (Fan & Song, 2014). The justification of this idea is based on the empirical results. These findings show when learning rate is very high, the algorithm do not converge, or it has too high penalty cost. On the other hand, when the learning rate is very low, the algorithm is very slow and ineffective. A usual approach is to start with a high learning rate and then decrease it over time. A better method is to adopt a learning rate that is increased slightly after each iteration where the cost is decreased, and it is decreased sharply when the cost rises.

Hence, we adopt a similar approach. More specifically, when the energy function (or cost function) decreases then we increase the learning rate in order to accelerate the convergence speed and when the same function is increased then we reduce the learning rate in oder to adjust the evolution process. We based our rate adjustments on the iteration number. Actually, this modification improves radically the quality of our solutions in local level, e.g. when we trigger the neural network from a good start point then the adaptive learning rate seems to perform perfectly.

Algorithm 2 Constraints Adjustment

SORT all items of $TempSchedule$ according to StartTime
for all items of Sorted $TempSchedule$ **do**
  if $TempSchedule_{i,j}.startime + TempSchedule_{i,j}.load > TempSchedule_{i,j+1}.startime$ **then**
    $TempSchedule_{i,j+1}.startime \leftarrow TempSchedule_{i,j}.startime + TempSchedule_{i,j}.load$
    INSERT $TempSchedule_{i,j}$ in $TempSchedule$ (using binary search)
  end if
  for all $TempSchedule_{x,y}$ involved in a Disjunctive Constraint with $TempSchedule_{i,j}$ **do**
    if $TempSchedule_{i,j}.startime + TempSchedule_{i,j}.load > TempSchedule_{x,y}.startime$ **then**
      $TempSchedule_{x,y}.startime \leftarrow TempSchedule_{i,j}.startime + TempSchedule_{i,j}.load$
      INSERT $TempSchedule_{x,y}$ in $TempSchedule$ (using binary search)
    else
      if $TempSchedule_{x,y}.startime + TempSchedule_{x,y}.load > TempSchedule_{i,j}.startime$ **then**
        $TempSchedule_{i,j}.startime \leftarrow TempSchedule_{x,y}.startime + TempSchedule_{x,y}.load$
        INSERT $TempSchedule_{i,j}$ in $TempSchedule$ (using binary search)
      end if
    end if
  end for
end for

Moreover, we associate the step of this adjustment with the resource density (density coefficient) of the benchmarks instances because the experimental results showed that the neural network evolution needs different rates regarding to their set of applied benchmark instances. That means that when there are many possible resource choices for an activity (e.g. rdata and vdata benchmark instances), the learning rate should be changed more drastically. So, we trigger the neural network evolution using an initial value for the learning rate (from 0.51 to 0.91) and then we permit the evolution to lead the rate's adjustment.

4.3. Adjustment Algorithm

One of the main concerns of the RNN's learning process is the feasibility of the derived schedules. Apparently, this is not guaranteed after each iteration of the learning process and an adjustment algorithm that maintains the feasibility of the schedule is required. The complexity of our problem makes the situation even worst because the schedule can be non feasible for two reasons, either a conjunctive or a disjunctive constraint is violated. So, it is rarely, that the derived schedule after each iteration remains feasible.

Recent research work (Kechadi et al., 2013), tried to solve this issue by applying two distinguished algorithms and facing separately the two types of constraint violation. This solution presents an obvious drawback in our case. More specifically, the algorithm that resolves the conjunctive constraint violation provokes more disjunctive constraint violations than the RNN's learning process and vice versa. Moreover, if the two algorithms use a shift approach in order to push every activity that has start time greater than the resolved activity every time a constraint violation is met, then the RNN's learning process actually leads to derived schedules that diverge significantly from the optimal or near optimal solution.

Our proposed solution tries to face simultaneously the two types of the constraint violation, conjunctive and disjunctive, constructing one algorithm that resolves the deficiencies of the past approaches. The main idea is to sort the schedule's activities according to their Start Time and then perform the core work of the adjustment. The proposed algorithm is analytically described in Algorithm 2.

First, we sort the array and then we start from the left, i.e. the activity which has the lowest Start Time and in each iteration of the learning process the algorithm checks if there is any violation either of a conjunctive constraint or disjunctive constraint. Every time a constraint violation is detected, it is

resolved by shifting an activity in order to ensure the feasibility. Moreover, after each correction the algorithm keeps the rest elements of the activities' array sorted by inserting the changed activities in the right position.

Finally, after the core algorithm's execution the schedule needs a left shift operation in order to eliminate the disturbing gaps between consecutive activities, that usually are created in each iteration of the RNN's learning process.

Algorithm 3 Tabu search like

for all $i \leq TabuArraySize$ **do**
  $TabuArray[i] \leftarrow MAXINT$
end for
CALCULATE $CurrentMakespan$
if $CurrentMakespan$ FOUND in $TabuArray$ **then**
  for all α_{ij} intact in last 2 iterations **do**
    FIND The first alternative resource r_m
    ASSIGN r_m to α_{ij}
  end for
  CALL Algorithm 2
  CALCULATE $CurrentMakespan$
else
  ADD $CurrentMakespan$ in $TabuArray$
end if

4.4. Avoiding local minima

The majority of the machine learning methods suffer from trapping in local minima (Reeves, 1995). This drawback led us to the development of a mechanism that eliminates or reduces the trapping effect. The main idea of our approach is based on a Tabu Search like algorithm. The Tabu array keeps stored the last derived from the neural network values of the Makespan objective. We have to underline that we don't need to store the whole derived schedules that give these values because this would increase significantly the total computation. Apparently, this is not a straightforward Tabu Search approach but rather a variant of this technique and surely it is faster than the original method.

More specifically, when an iteration of the RNN's learning process is completed the Makespan's value is calculated, and then the algorithm searches

the Tabu array structure in order to find if the new derived value is already stored in this array. If the value is not found in the Tabu array then it is stored as the last derived value by shifting the rest values, i.e. the first value of the array is dropped.

If the Makespan's new value is found in the Tabu array then the algorithm has to act, in order to escape from the possible local minimum. Moreover, this algorithm has to deal with the observed ping-pong effect that is displayed when is applied in some benchmark instances. This is also the main cause why we use Tabu Search algorithm instead of a simple algorithm that deals only with the local minima.

However, there is a main issue that must be resolved, in order to help the solving method escaping from the local minima or the ping-pong area. There are two options for this, the first is to adopt a perturbation algorithm such as the algorithm proposed by (Kechadi et al., 2013) or to invent an algorithm that will change the initial resources' assignment of some activities. The latter option can be performed more efficiently on benchmark instances that offer more than one resource choices for each activity. We have to underline that the Algorithm 3 is executed in every iteration of the RNN's learning process and it increases the total time of the convergence. The brief description of the second approach, i.e. the alteration of the assignments is as follows: the algorithm finds an activity that is intact after the last two iterations of the RNN's learning process and then it finds an alternative resource that is valid for assignment to this activity. Intact activity is the activity which retains the same start time and same resource's assignment after an iteration of the learning process.

The alteration process consists of five steps:

- Find an intact activity.
- Assign the alternative resource r_m to this activity.
- Change the constraints' structure, in order to activate the new constraints and deactivate the old ones (constraints' refreshment).
- Call the adjustment Algorithm 2, in order to keep the derived schedule feasible.

The whole process, i.e. the RNN's learning process is terminated when the Makespan is near to the known optimal solution or near optimal solution

(optimal solution's area). An alternative termination condition can be a given threshold of the processing time or a specific number of iterations.

The overall approach is illustrated graphically in Figure 5. Initially, two schedules are constructed and then the Genetic Algorithm 1 derives eight additional schedules. Each of these 10 schedules feeds a correspondent RNN which starts its learning process using the improvement Algorithms 2 and 3. Finally, the best of the ten derived schedules, the outcome of the RNNs, is the final solution of the problem.

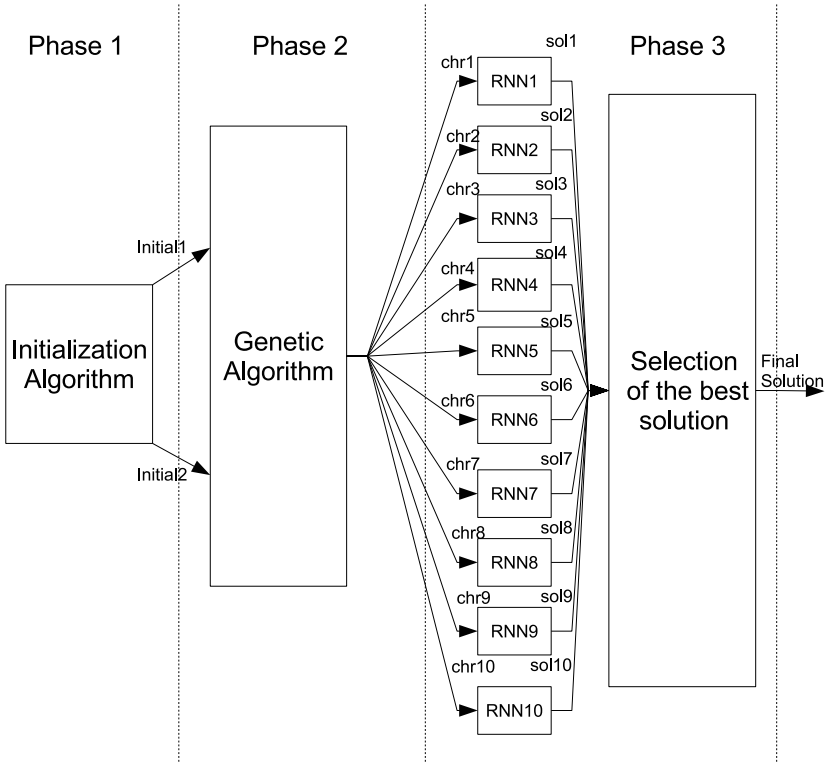

Figure 5: The diagram of the overall proposed solution approach.

5. Experimental Results

The proposed approach is performed on the benchmark instances described analytically in section 3 and the derived results are presented in

tables 2, 3, 4, and 5. Although the algorithms and the overall simulator are written in C programming language, the processing of the results was implemented in R language. The values of the parameters that have been used are as follows:

- **Penalty factors** $K_o = 0.06$ and initial **learning rates** $\mu_o \in [0.51, 0.96]$.
- **Increase step** of $\mu_o = \mu + iter\_no/ITERATIONS$ and **Decrease step** of $\mu_o = \mu - (ITERATIONS - iter\_no)/ITERATIONS$.
- The maximum number of iterations $ITERATIONS = 100$.

The following paragraphs present briefly the results observed from each table and its correspondent set of benchmark instances. The columns and their meaning of every table are as follows: the Benchmark Instance (**Instance**), the obtained value of the Makespan (**MakeSpan**), the percent of variance between the Makespan and the known optimal or near optimal (**Var%**), the obtained value of the Total Weighted Tardiness (**WTardiness**), the initial Learning Rate (**LearningRate**) that gave the best result, the execution time (**ExecTime**) and the Chromosome number (**Chromosome**), i.e. the initial schedule that leaded to this result.

Sdata set. Generally, our approach performs very effectively on the sdata set of instances (the sdata is the original set of the typical JSSP benchmark instances, i.e. for each activity there is only one resource that can be assigned). More specifically, the observed results in table 2 reveal that in the majority of the instances (29 out of 31) the value of the Makespan objective reached the optimal or is quite close to this. Moreover, these values are obtained within the minimum of the energy function and low values of the second objective, Total Weighted Tardiness. On the other hand, our approach execution failed to converge in few instances (2 out of 31).

Naturally, the GA approach does not offer any benefit to the sdata benchmark instances set because this set is the naive JSSP benchmark set where each activity has a unique resource that can be assigned and the there is not any possible chance for mutation. So, the improvement that is observed in many instances is accomplished exclusively by the adaptive learning rate application.

Edata set. The overall proposed approach operates almost perfectly on the edata set of benchmark instances (the edata is the set of benchmark instances that offer to activities one or two possible resources for assignment).

Table 2: Results for SDATA benchmark instances

	Instance	MakeSpan	Var%	WTardiness	LearningRate	ExecTime	Chromosome
1	car1	5651.00	12.91	2683.00	0.61	0.15	1
2	car2	5929.00	0.00	0.00	0.51	0.15	1
3	car3	5633.00	0.64	1231.00	0.51	0.19	2
4	car4	6674.00	2.46	1630.00	0.51	0.21	1
5	car5	6925.00	41.07	0.00	0.56	0.15	1
6	car6	7349.00	33.96	620.00	0.51	0.24	2
7	car7	6821.00	61.79	116.00	0.61	0.00	2
8	car8	6120.00	32.67	0.00	0.56	0.17	2
9	la01	666.00	0.00	0.00	0.51	0.12	1
10	la02	655.00	0.00	109.00	0.66	0.10	1
11	la03	597.00	0.00	265.00	0.51	0.08	1
12	la04	590.00	0.00	8.00	0.56	0.09	2
13	la05	593.00	0.00	136.00	0.51	0.12	1
14	la06	926.00	0.00	288.00	0.51	0.20	1
15	la07	890.00	0.00	51.00	0.51	0.12	1
16	la08	863.00	0.00	0.00	0.51	0.22	1
17	la09	951.00	0.00	297.00	0.61	0.19	1
18	la10	958.00	0.00	384.00	0.51	0.27	1
19	la11	1222.00	0.00	696.00	0.51	0.53	1
20	la12	1039.00	0.00	787.00	0.56	0.54	1
21	la13	1150.00	0.00	372.00	0.51	0.41	1
22	la14	1292.00	0.00	54.00	0.51	0.50	1
23	la15	1207.00	0.00	252.00	0.51	0.87	2
24	la16	2495.00	164.02	3331.00	0.66	0.01	1
25	la17	784.00	0.00	52.00	0.51	0.17	1
26	la18	2939.00	246.58	5036.00	0.51	0.00	1
27	la19	842.00	0.00	0.00	0.51	0.28	1
28	la20	902.00	0.00	0.00	0.56	0.23	2
29	mt06	76.00	38.18	28.00	0.96	0.00	2
30	mt10	871.00	0.00	0.00	0.51	0.49	1
31	mt20	1088.00	0.00	658.00	0.51	1.18	1

In table 3, it is easy for someone to see that in most cases (30 out of 31) the value of the Makespan objective reached the optimal or is quite close to it. Moreover, these values are obtained within the minimum of the energy function and low values of the second objective, Total Weighted Tardiness. However, our approach execution failed to converge in few instances (1 out of 31).

The experimental results on this set of benchmark instances show that the impact of GA application was moderate or sometimes almost insignificant. The main cause of this finding is that in this benchmark set there are only one or two possible resources for each activity. Therefore, there are very few possible gene changes in the mutation process, in order to increase the diversity of the two initial solutions that feed the Genetic Algorithm. In other words, the adoption of a GA approach for this benchmark set has not improved significantly the quality of the final solution.

However, the neural network convergence operates sufficiently for the most cases giving final solutions very close to the known optimal or near optimal solution except TWTla16, TWTla18, and TWTla20. Obviously, the most complex instances (TWTla21-TWTla40) cannot give competitive

Table 3: Results for EDATA benchmark instances

	Instance	MakeSpan	Var%	WTardiness	LearningRate	ExecTime	Chromosome
1	car1	6176.00	0.00	1750.00	0.61	0.22	1
2	car2	6455.00	0.00	1938.00	0.51	0.02	3
3	car3	6856.00	0.00	2308.00	0.51	0.13	1
4	car4	7789.00	0.00	8825.00	0.61	0.10	2
5	car5	7229.00	0.00	0.00	0.51	0.00	3
6	car6	8478.00	0.00	2184.00	0.61	0.23	1
7	car7	6123.00	0.00	1022.00	0.56	0.01	1
8	car8	7689.00	0.00	3457.00	0.51	0.15	1
9	la01	665.00	9.20	54.00	0.76	0.09	1
10	la02	686.00	4.73	252.00	0.66	0.13	2
11	la03	550.00	0.00	214.00	0.51	0.15	2
12	la04	568.00	0.00	0.00	0.56	0.21	1
13	la05	552.00	9.74	107.00	0.86	0.18	2
14	la06	833.00	0.00	0.00	0.51	0.01	3
15	la07	762.00	0.00	184.00	0.51	0.00	3
16	la08	1122.00	32.78	972.00	0.51	0.41	2
17	la09	878.00	0.00	402.00	0.51	0.42	2
18	la10	866.00	0.00	207.00	0.66	0.38	1
19	la11	1192.00	9.66	814.00	0.56	0.02	4
20	la12	1155.00	20.31	0.00	0.51	0.01	5
21	la13	1377.00	30.77	2017.00	0.51	0.73	2
22	la14	1123.00	0.00	0.00	0.51	0.02	3
23	la15	1431.00	28.80	735.00	0.51	0.81	1
24	la16	948.00	6.28	0.00	0.66	0.41	2
25	la17	827.00	16.97	57.00	0.56	0.66	1
26	la18	2584.00	206.89	2612.00	0.51	0.01	3
27	la19	796.00	0.00	60.00	0.61	0.44	1
28	la20	1213.00	41.54	246.00	0.61	0.15	2
29	mt06	55.00	0.00	14.00	0.71	0.00	2
30	mt10	957.00	9.87	34.00	0.66	0.57	1
31	mt20	1373.00	26.19	2984.00	0.51	0.65	1

solutions and they converge very far away from the optimal or near optimal. In conclusion, in the edata set the solution's improvement that is observed in many instances is accomplished exclusively by the adaptive learning rate application and rarely by the GA's application.

Rdata set. Regarding to the rdata set of benchmark instances (the rdata is the set of benchmark instances whose activities are offered more than two possible resources for assignment), our approach gives very promising results for many instances although they cannot be compared with the above sets' results. Observing table 4, it is easy for someone to see that in some cases the makespan objective obtained values quite away from the optimal but we have to bear in mind that our goal is to minimise simultaneously two objective functions. Moreover, the second objective, Total Weighted Tardiness, obtained very low values and the convergence is accomplished in few iterations. However, there are some instances that gave non competitive results and they require more investigation (8 out of 31 instances).

In this set of benchmark instances, the derived solutions are significantly improved by the GA application in the Phase 2 of the proposed solution method. Apparently, this set of instances provides more resources as assign-

Table 4: Results for RDATA benchmark instances

	Instance	MakeSpan	Var%	WTardiness	LearningRate	ExecTime	Chromosome
1	car1	7702.00	52.30	1440.00	0.51	0.01	3
2	car2	5987.00	0.00	299.00	0.51	0.23	3
3	car3	5626.00	0.00	0.00	0.51	0.90	4
4	car4	6518.00	0.00	0.00	0.51	0.33	5
5	car5	5764.00	0.00	0.00	0.51	0.01	3
6	car6	6147.00	0.00	0.00	0.51	0.01	3
7	car7	10295.00	132.29	5481.00	0.51	0.34	1
8	car8	5692.00	0.00	0.00	0.51	0.26	3
9	la01	1247.00	118.77	876.00	0.56	0.01	4
10	la02	738.00	39.51	123.00	0.51	0.08	3
11	la03	477.00	0.00	0.00	0.51	0.00	3
12	la04	502.00	0.00	0.00	0.51	0.07	3
13	la05	601.00	31.51	0.00	0.51	0.01	3
14	la06	799.00	0.00	113.00	0.51	0.17	3
15	la07	749.00	0.00	34.00	0.51	0.01	3
16	la08	1416.00	85.10	595.00	0.51	0.01	3
17	la09	853.00	0.00	0.00	0.51	0.15	3
18	la10	804.00	0.00	0.00	0.51	0.02	5
19	la11	1071.00	0.00	0.00	0.51	0.02	9
20	la12	1393.00	48.82	121.00	0.56	0.62	5
21	la13	1038.00	0.00	29.00	0.51	0.05	3
22	la14	1070.00	0.00	0.00	0.51	0.09	3
23	la15	1149.00	5.51	185.00	0.51	0.61	7
24	la16	1638.00	128.45	563.00	0.51	0.16	3
25	la17	1723.00	166.72	1981.00	0.51	0.01	5
26	la18	1524.00	128.83	929.00	0.51	0.31	3
27	la19	844.00	30.45	0.00	0.61	0.28	5
28	la20	2445.00	223.41	1460.00	0.51	0.51	3
29	mt06	52.00	10.64	6.00	0.51	0.00	3
30	mt10	1113.00	63.92	430.00	0.56	0.03	3
31	mt20	1022.00	0.00	0.00	0.51	1.42	4

ment options to activities and consequently the mutation process generates offspring with wider diversity than those given by the sdata and edata sets. This conclusion, according to the experimental results, is true for the low complexity instances. Unfortunately, we cannot assert the same conclusion for the rest of the instances where the observed improvement does not guarantee high quality final solutions. Probably, a more sophisticated mutation process would amend this issue in future work.

Vdata set. It is very stunning to find that the overall proposed approach performs adequately on the vdata set of benchmark instances although this specific set is the most complicated one because for each activity assignment there are too many alternative available resources and consequently there are too many disjunctive constraints. More specifically, in table 5 it is easily observed that in more than two thirds of the instances the Makespan objective obtained values quite close to the optimal (21 out of 31 instances).

On the other hand, the neural network did not converge sufficiently in 10 out of 31 instances, but this can be explained because our ultimate goal is to minimise simultaneously two objective functions. Moreover, the second objective, Total Weighted Tardiness, obtained very low values and the con-

Table 5: Results for VDATA benchmark instances

	Instance	MakeSpan	Var%	WTardiness	LearningRate	ExecTime	Chromosome
1	car1	5013.00	0.00	0.00	0.51	0.50	3
2	car2	6078.00	2.50	339.00	0.56	0.32	4
3	car3	5600.00	0.00	0.00	0.51	0.87	3
4	car4	6517.00	0.00	0.00	0.56	0.56	4
5	car5	4932.00	0.00	0.00	0.51	0.03	5
6	car6	5486.00	0.00	0.00	0.51	0.13	3
7	car7	29737.00	594.63	79402.00	0.56	0.01	1
8	car8	13387.00	190.20	8504.00	0.71	3.09	3
9	la01	1016.00	78.25	1321.00	0.51	0.00	4
10	la02	529.00	0.00	35.00	0.51	0.44	3
11	la03	3793.00	695.18	16163.00	0.51	0.00	2
12	la04	502.00	0.00	0.00	0.51	0.14	3
13	la05	766.00	67.61	103.00	0.56	0.01	3
14	la06	799.00	0.00	0.00	0.51	0.21	3
15	la07	749.00	0.00	0.00	0.51	0.04	3
16	la08	1480.00	93.46	591.00	0.51	0.49	3
17	la09	2162.00	153.46	6580.00	0.51	0.01	4
18	la10	1534.00	90.80	2595.00	0.51	0.02	5
19	la11	1071.00	0.00	0.00	0.51	0.15	4
20	la12	1461.00	56.09	0.00	0.51	1.42	4
21	la13	2531.00	143.83	7517.00	0.51	0.04	5
22	la14	1070.00	0.00	0.00	0.51	0.73	3
23	la15	1089.00	0.00	255.00	0.51	8.74	3
24	la16	2087.00	191.07	1133.00	0.51	2.50	3
25	la17	6538.00	912.07	11233.00	0.51	0.06	3
26	la18	13676.00	1962.75	88355.00	0.86	0.17	2
27	la19	617.00	0.00	0.00	0.51	67.73	3
28	la20	12063.00	1495.63	19996.00	0.51	0.12	3
29	mt06	47.00	0.00	0.00	0.51	0.02	3
30	mt10	2839.00	333.44	2168.00	0.61	8.06	3
31	mt20	1022.00	0.00	0.00	0.51	5.29	3

vergence is accomplished in quite few iterations. However, there are many instances that gave non competitive results and they require further investigation. Obviously, the Genetic Algorithm improved the final solution but not so significantly as it happens in the rdata set.

6. Conclusion

In this work, we presented a combination of intelligent methods into an integrated approach that address effectively a multi-objective scheduling problem, the MPM Job Shop Scheduling Problem. By developing a model based on JSSP formulation, we utilized a Genetic Algorithm that enhances the diversity of the initial schedules, which in their turn feed Recurrent Neural Networks, in order to find the best combination of the two objectives, i.e. the Makespan and the Total Weighted Tardiness. Moreover, we accelerate the RNN's convergence speed by using three improvement techniques, an adaptive learning rate, an adjustment algorithm, and Tabu Search like algorithm. Finally, we presented some of the experimental findings by applying the set of the above techniques on benchmark instances that are partially our construction in order to suit the total weighted tardiness objective.

According to these results, the RNN learns quite effectively but in some cases, the results initially were disappointing. Hence, additional techniques and suitable adjustments employed, in order to enhance the effectiveness of the proposed method. More specifically, an adaptive learning rate policy improved the quality and the speed of convergence while an adjustment algorithm ensured the feasibility of the derived schedules, and a Tabu Search like technique contributed to the avoidance of the local minima. Eventually, the bundle of these amendments enhanced significantly the RNN's efficiency and enlarged the set of benchmark instances which give competitive solutions.

As future work, we can establish this set as set of benchmark instances for the multi-objective MPM JSSP problem and this can enhance the innovative contribution of the work. We have also to underline that we don't use very complicated benchmark instances (e.g. la21-la40) because they may represent extreme cases of real projects.

References

Adibi, M. A., Zandieh, M. & Amiri, M. (2010), 'Multi-objective scheduling of dynamic job shop using variable neighborhood search', *Expert Systems with Applications* **37**(1), 282–287.

Agarwal, A., Colak, S. & Erenguc, S. (2011), 'A neurogenetic approach for the resource-constrained project scheduling problem', *Computers and Operations Research* **38**(1), 44–50.

Bernd Jurisch (1992), Scheduling Jobs in Shops with Multi-Purpose Machines, PhD Thesis, University of Osnabrueck.

Browning, T. R. & Yassine, A. A. (2010), 'Resource-constrained multi-project scheduling: Priority rule performance revisited', *International Journal of Production Economics* **126**(2), 212–228.

Brucker, P., Drexl, A., Mohring, R., Neumann, K. & Pesch, E. (1999), 'Resource-constrained project scheduling: Notation, classification, models, and methods', *European Journal of Operational Research* **112**(1), 3–41.

Cavin, L., Fischer, U., Glover, F. & Hungerbühler, K. (2004), 'Multi-objective process design in multi-purpose batch plants using a Tabu Search optimization algorithm', *Computers and Chemical Engineering* **28**(4), 459–478.

Chelouah, R. & Siarry, P. (2000), 'Tabu search applied to global optimization', *European Journal of Operational Research* **123**(2), 256–270.

Demirkol, E., Mehta, S. & Uzsoy, R. (1998), 'Benchmarks for shop scheduling problems', *European Journal of Operational Research* **109**(1), 137–141.

Fan, H. & Song, Q. (2014), 'A linear recurrent kernel online learning algorithm with sparse updates', *Neural Networks* **50**, 142153.

Graham R. L., Lawler E. L., Lenstra J.K. & Rinnooy Kan A.H.G. (1979), Optimization and approximation in deterministic suquencing and scheduling: a survey, *in* 'Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium.', Vol. 5, Elsevier, pp. 287–326.

Kechadi, M.-T., Low, K. & Goncalves, G. (2013), 'Recurrent neural network approach for cyclic job shop scheduling problem', *Journal of Manufacturing Systems* **32**(4), 689–699.

Lawrence, S. (1994), *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement)*, Graduate School of Industrial Administration,Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Reeves, C. R. (1995), *Modern Heuristic Techniques for Combinatorial Problems*, McGraw-Hill.

Schwindt, C. (2005), *Project management and resource allocation*, Springer-Verlag, Heidelberg.

Trojet, M., H'Mida, F. & Lopez, P. (2010), 'Project scheduling under resource constraints: Application of the cumulative global constraint in a decision support framework', *Computers & Industrial Engineering*.

Tselios, D., Savvas, I. & Kechadi, M.-T. (2013*a*), 'Multiple project portfolio scheduling using recurrent neural networks', *International Journal of Simulation and Process Modelling* **8**(4), 227–240.

Tselios, D., Savvas, I. K. & Kechadi, T. M. (2013*b*), 'MPM job shop scheduling problem: a bi-objective approach', *International Journal of Simulation Systems, Science and Technology* **14**(5), –.

Zhang, R., Xu, Z.-B., Huang, G.-B. & Wang, D. (2012), 'Global convergence of online BP training with dynamic learning rate', *IEEE Transactions on Neural Networks and Learning Systems* **23**(2), 330–341.