Chapter 12

RESOURCE CONSTRAINED PROJECT SCHEDULING: A HYBRID NEURAL APPROACH

Selcuk Colak, Anurag Agarwal, Selcuk S. Erenguc
Department of Decision and Information Sciences, Warrington College of Business Administration, University of Florida, Gainesville, FL 32611, USA
scolak@ufl.edu, anurag.agarwal@cba.ufl.edu, selcuk.erenguc@cba.ufl.edu

Abstract This study proposes, develops and tests a hybrid neural approach (HNA) for the resource constrained project scheduling problem. The approach is a hybrid of the adaptive-learning approach (ALA) for serial schedule generation and the augmented neural network (AugNN) approach for parallel schedule generation. Both these approaches are based on the principles of neural networks and are very different from Hopfield networks. In the ALA approach, weighted processing times are used instead of the original processing times and a learning approach is used to adjust weights. In the AugNN approach, traditional neural networks are augmented in a manner that allows embedding of domain and problem-specific knowledge. The network architecture is problem specific and a set of complex neural functions are used to (i) capture the constraints of the problem and (ii) apply a priority rule-based heuristic. We further show how forward-backward improvement can be integrated within the HNA framework to improve results. We empirically test our approach on benchmark problems of size J30, J60 and J120 from PSPLIB. Our results are extremely competitive with existing techniques such as genetic algorithms, simulated annealing, tabu search and sampling.

Keywords: Project Management, Resource Constrained Project Scheduling, Neural Networks, Heuristics

12.1 Introduction

The resource-constrained project scheduling problem (RCPSP) is a well-known NP-Hard scheduling problem (Blazewicz et al (1983)). It is a classical problem in operations research with broad applicability in project management and production scheduling. It involves minimizing the makespan of a project

by scheduling its activities which are subject to precedence and resource constraints. The amounts of available resources are fixed and known in advance. Resource requirements and processing times for each activity are deterministic and also known in advance and preemption of activities is not allowed. This problem has received the attention of many researchers for well over four decades. One of the recent research focuses in this area has been towards developing new metaheuristic approaches using artificial intelligence and/or biologically-inspired techniques. For solving this problem, two schedule generation schemes are commonly used – serial and parallel. In this work, we propose, develop and test a new hybrid metaheuristic approach based on the principles of neural networks. We use adaptive-learning approach (ALA) for serial and augmented-neural-network approach (AugNN) for parallel schedule generation scheme. We call our approach the hybrid-neural approach (HNA).

In the adaptive-learning approach (Agarwal et al (2005)), weighted processing times are used instead of the given processing times. Well-known heuristics are applied using these weighted processing times. An intelligent perturbation strategy used to adjust the weights allows non-deterministic local search. The AugNN approach was first applied to parallel schedule generation in the task-scheduling problem by Agarwal et al (2003). With suitable modifications, the AugNN approach can be applied to the parallel generation scheme for the RCPSP. The AugNN approach is quite different from the Hopfield network approach which has been applied to the traveling-salesman problem (Hopfield and Tank (1985) and job-shop scheduling (Sabuncuoglu and Gurgun (1996), Foo and Takefuji (1988)). In the AugNN approach, the traditional neural network is augmented to allow embedding of domain and problem-specific knowledge. The network architecture is designed to be problem specific; instead of the standard 3-layered network, it is a p-layered network, where p depends on the problem structure. Details will be explained in Section 16.4. Further, in the AugNN approach, the input, activation and output functions are complex functions, designed to (i) enforce the problem constraints, and (ii) apply a known priority heuristic. The AugNN approach, thus, allows incorporation of domain and problem-specific knowledge and affords the advantages of both the heuristic and iterative approaches. In this study, forward-backward improvement steps (Tormos and Lova (2001), Valls et al (2005) are also integrated within this framework of hybrid-neural approach.

We implement and test our proposed hybrid-neural approach on some well-known RCPS benchmark problem instances in the literature. Our results are very competitive with those of other techniques. Given that this approach is relatively new, it seems to hold a lot of promise; perhaps in future studies, it can be used in conjunction with other successful techniques, such as genetic algorithms, scatter search etc. to give improved results.

The rest of the paper is organized as follows. Section 16.2 presents a literature review for the RCPSP. We discuss how ALA is applied to the serial schedule generation problem in Section 16.3. The details of AugNN formulation for solving the parallel schedule generation for the RCPSP are given in Section 16.4. In Section 16.5, computational results are presented and discussed. Finally, Section 15.6 provides a summary of the paper and discusses future research ideas.

12.2 Literature review

The research literature for the RCPSP is quite large. We refer the readers to the review papers by Icmeli et al (1993), Ozdamar and Ulusoy (1995), Herroelen et al (1998), Brucker et al (1999), Hartmann and Kolisch (2000), Kolisch and Padman (2001), Kolisch and Hartmann (2005).

The various exact methods applied to the RCPSP can be classified into three categories: dynamic programming, zero-one programming and implicit enumeration with branch and bound. Pritsker et al (1969), Patterson and Huber (1974), Patterson and Roth (1976) proposed zero-one programming methods. Exact approaches based on implicit enumeration with branch and bound have been widely used: Davis and Heidorn (1971), Talbot and Patterson (1978), Christofides et al (1987), Demeulemeester and Herroelen (1992), Demeulemeester and Herroelen (1997), Brucker et al (1998), Mingozzi et al (1998), Dorndorf et al (2000). Blazewicz et al (1983) showed that the RCPSP is a generalization of the well-known job-shop-scheduling problem and is NP-Hard. While exact solution methods are able to solve smaller problems, heuristic and metaheuristic approaches are needed for larger problem instances.

Priority-rule based heuristics combine one or more priority rules and schedule-generation schemes (serial, parallel or both) in order to construct one or more schedules (Hartmann and Kolisch (2000)). If only one schedule is generated, it is called a single pass method and if more than one schedule is generated, it is called an X-pass (or multi-pass) method. Some of the well-known priority rules are LFT (Latest Finish Time), EST (Earliest Start Time) and MTS (Most Total Successor). Although, priority-rule based heuristics are easy to implement and fast in terms of the computational effort, they are not very effective with respect to the average deviation from the optimal solution. A variety of priority single-pass methods have been widely used to solve the RCPSP: Davis and Patterson (1975), Cooper (1976), Alvares-Valdes and Tamarit (1989), Boctor (1990), Ozdamar and Ulusoy (1994), Kolisch (1996a), Kolisch (1996b). Multi-pass methods can be categorized as multi-priority rule methods and sampling methods. Multi-priority rule methods combine the schedule generation scheme with a different priority rule at each iteration: Ulusoy and Ozdamar (1989), Boctor (1990), Thomas and Salhi (1998). Sampling methods

use a serial generation scheme and a priority rule to obtain the first schedule. Then they bias the order obtained by the priority rule by using a random device: Cooper (1976), Alvares-Valdes and Tamarit (1989), Drexl (1991), Kolisch (1996a), Kolisch (1996b), Kolisch and Drexl (1996), Schirmer and Riesenberg (1998), Schirmer (2000).

Many metaheuristic methods, such as genetic algorithms (GA), simulated annealing (SA), tabu search (TS), and ant colonies (AC), have been applied to solve the RCPSP. Metaheuristics based on GA are the most common: Leon and Ramamoorthy (1995), Lee and Kim (1996), Hartmann (1998), Hartmann (2002), Alcaraz and Maroto (2001), Coelho and Tavares (2003), Hindi et al (2002), Toklu (2002), Valls et al (2003). Simulated annealing algorithms which can handle non-preemptive resource constrained project scheduling problem are presented by Boctor (1996), Cho and Kim (1997), Bouleimen and Lecocq (2003). Tabu search based metaheuristics are proposed by Pinson et al (1994) and Baar et al (1997), Nonobe and Ibaraki (2002) and Thomas and Salhi (1998). Merkle et al (2002) proposed an ant colony approach to the RCPSP.

In addition to applying these heuristics and metaheuristics, forward-backward improvement (FBI) steps are suggested by Tormos and Lova (2001), Tormos and Lova (2003) and Valls et al (2005). This step is also called double justification technique. In FBI, a given schedule is compressed by eliminating unnecessary pockets of slack on a Gantt Chart.

Surprisingly, neural network (NN) based techniques have not been applied to the RCPSP to the best of our knowledge. NN based approach has been applied to the job-shop scheduling problem (Foo and Takefuji (1988), and Sabuncuoglu and Gurgun (1996)), and the traveling salesman problem (Hopfield and Tank (1985)). Their approach is based on Hopfield networks. While this approach worked for smaller problem instances (up to 5x5), it failed to provide good solutions in reasonable time, for larger problem instances such as (10x10). Agarwal et al (2003) proposed a different kind of approach for using neural networks, called the AugNN approach, for solving task-scheduling problems. The performance of this alternative NN approach does not deteriorate with larger problem size.

12.3 Adaptive learning approach for serial schedule generation

As mentioned in Section 1, two types of schedule generation schemes are used in RCPSP, viz., serial and parallel. In the serial scheduling scheme, a priority list of activities is determined at time zero. This list is based on some heuristic such as latest finish time (LFT). The ordering of activities in a given priority list must, of course, follow precedence constraints, but is independent of the resource constraints. Given a priority list, activities are scheduled in the

given order at the earliest possible clock time at which the precedence constraints are satisfied and the resources are available. This type of scheduling is similar to the permutation flow-shop scheduling in which the order of jobs is fixed a priori and scheduling occurs at the earliest possible clock time depending on resource availability and precedence constraints. Agarwal et al (2005) applied an adaptive learning approach to the permutation flow-shop problem. Due to the similarities between these two problems, we apply the ALA approach to the serial schedule generation.

The ALA is a non-deterministic local-search approach based on neural-networks principles. In this approach, processing times of activities are parameterized using a weight factor. The problem of optimally scheduling a given project is then posed as the problem of finding the optimal set of weights in the weight search space, similar to the way non-linear mapping functions are determined in neural networks. Reinforcement and backtracking techniques are applied as part of weight modification strategies.

Notation used:

A	:	Set of activities = $\{1, .., n\}$
A_j	:	j^{th} activity node, $j \in A$
z	:	Current iteration
PT_j	:	Processing time of activity j
W_j	:	Weight associated with the Activity A_j
WPT_j	:	Weighted processing time of Activity A_j
EST_j	:	Earliest start time of activity j
LST_j	:	Latest start time of activity j
Z_{max}	:	Max number of iterations
MS_z	:	Makespan in the z^{th} iteration
RF	:	Reinforcement factor
$TINI$	:	Tolerate iterations with no improvement
α	:	Learning rate
BMS	:	Best makespan
BW_j	:	Best weights

Step 1 : Initialization

Initialize $\forall j \in A \quad W_j = 1$

Initialize the iteration counter z to 1.

Step 2 : Calculate weighted processing times

Calculate $\forall j \in A \quad WPT_j = W_j * PT_j$

Step 3 : Determine priority list

Determine the priority list using a heuristic such as earliest start time (EST) or latest finish time (LFT), where EST or LFT are calculated using WPT_j instead of PT_j.

Step 4 : *Determine makespan*

Find a feasible schedule using the priority list. In other words, schedule each activity at its earliest possible time given precedence and resource constraints. This schedule gives us the makespan MS_z.

Step 5 : *Apply learning strategy and modify weights*

a. If MS_z is the best makespan so far, save the current weights as best weights (BW_j) and the makespan as the best makespan (BMS).

b. If $z = Z_{max}$, go to Step 7.

c. If $z > 1$, and if an improvement occurs, reinforce the weights as follows:
$(W_j)_z = (W_j)_z + RF * ((W_j)_z - (W_j)_{z-1})$.
If no improvement occurs in this iteration, continue.

d. If $z > TINI$ and if no improvement has occurred in the past *TINI* iterations then
Set $W_j = BW_j$

e. Modify the weights using the following strategy:
Generate a random number RND between 0 and 1 using uniform distribution.
If RND > 0.5 then $(W_j)_{z+1}$ = $(W_j)_z$ + RND*α *P_j
If RND $<= 0.5$ then $(W_j)_{z+1}$ = $(W_j)_z$ - RND*α *P_j

Step 6 : *Next iteration.*

Increment z by one and go to step 2.

Step 7 : *Display Solution*

The BMS is the solution. Take the BWj and generate the schedule using the heuristic.

The learning rate (α) used in Step 5e determines the degree of weight change per iteration. A higher rate leads to a greater change and vice versa. One could therefore control the granularity of the search by varying α. The learning rate should neither be too low nor too high. A low α will slow down convergence and make it difficult to jump local minima, while a high α will render the search too erratic or volatile to afford convergence. With some empirical trial and error,. we found that a rate of 0.001 worked well for all the problems.

The reinforcement factor, RF, used in Step 5c is used to reinforce weights during iterations that show improved results. Intuitively, such reinforcement learning, common in neural networks, helps the search process by allowing the search to explore the newly discovered good neighborhood for a few extra iterations. Empirically, we found that an RF value of 2 gave better results than other RF values. Backtracking used in step 5d is used to backtrack to the previous best set of weights if no improvement has been found in a given number of iterations.

12.4 AUGNN framework for parallel schedule generation

In parallel schedule generation, the order in which the activities are going to be scheduled is not decided at time zero. The scheduling decisions are made on a clock timer, at times when activities can start and resources are available. Agarwal et al (2003) applied the AugNN approach for parallel schedule generation for the task scheduling problem which is a special case of RCPSP. In task scheduling, there is only one type of resource and each task needs only one unit of that resource type. We extend Agarwal et al's AugNN formulation of task scheduling problem to generate parallel schedules for RCPSP in which there are multiple resources and each activity needs multiple units of each resource. We describe the AugNN framework with the help of a simple RCPS problem shown in Figure 16.1. In this problem, there are six activities plus two zero-time dummy activities for the initial and final activities. There are three renewable resource types R1, R2 and R3. Each activity's duration and resource requirements are also shown in Figure 16.1. In this problem, the longest path (in terms of number of activities) has 5 activities – A1, A4, A6 or A2, A5, A6 plus the two dummy activities.

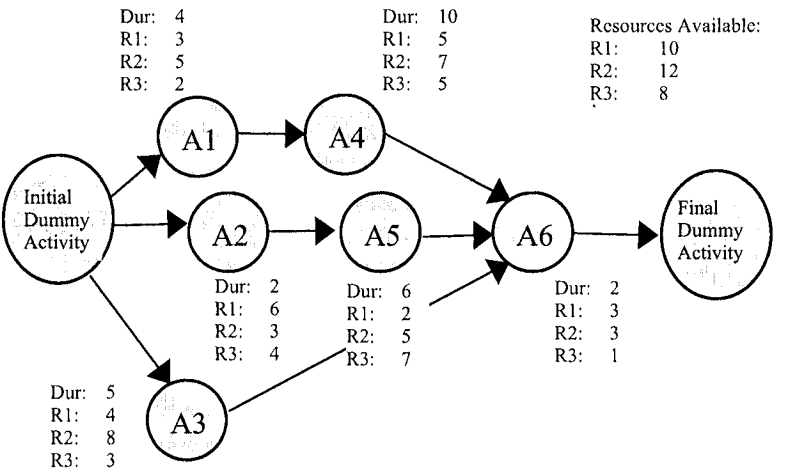

Figure 12.1. Example RCPS Problem

In the AugNN approach, the project graph of Figure 16.1 is framed as an n-layered neural network as shown in Figure 16.2. n, in this case is 8, which is 2(5-1), because 5 is the number of activities on the longest path. The set of activities at each level are placed in an activity layer. The p^{th} activity layer is p activities removed from the initial dummy activity. Each activity layer, except the dummy activity layers, is followed by a resource layer. The resource layer represents all the available resources. Input, activation and output functions for activity and resource layers are designed to capture the constraints of the problem. We will briefly describe the purpose of these functions.

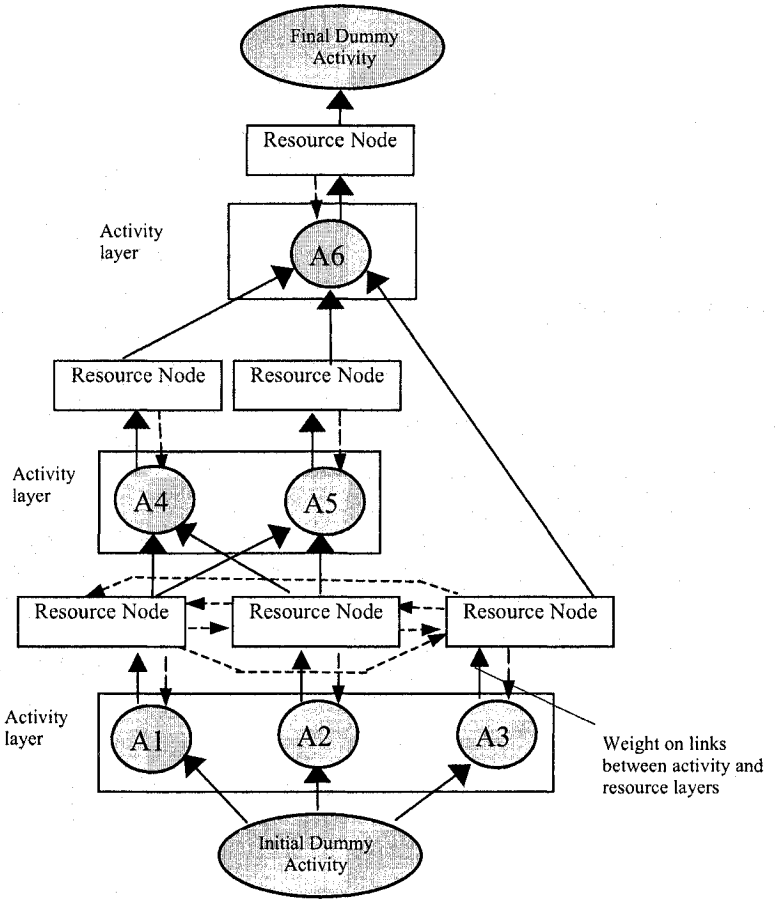

Figure 12.2. AugNN Representation of the Example Problem

Input Functions

The activity nodes get their input from the initial dummy node and resource layers. These functions are used to enforce the precedence constraints. When an activity node receives as many signals as the number of preceding activities, the activity is considered ready to be assigned.

The resource layer gets its input from the activity nodes preceding it. When it gets an input, it knows that the activity is ready to start and that resources, if available, can be assigned.

Activation Functions

The activation functions of the activity nodes maintain the state of each activity. The activation functions of the resource layer keep track of resource availability and assignment of resources to activities. The priority rule is applied through these activation functions.

Output Functions

The output function of the activity node sends a signal to the resource layer when it is ready to be assigned. The output function of the resource layer signals the end of the activity.

12.4.1 Mathematical Formulation and Algorithm Details

Notation

n	:	Number of activities
r	:	Number of resource types
A	:	Set of activities = $\{1, \ldots, n\}$
R	:	Set of resource types = $\{1, \ldots, r\}$
r_{ik}	:	Amount of resource k required by activity i, $k \in R$, $i \in A$
b_k	:	Total availability of resource type k
k	:	Current iteration
A_j	:	j^{th} activity node, $j \in A$
RL_j	:	Node for resource layer connected from $A_j, j \in A$
TS_j	:	Total number of successors of $A_j, j \in A$
$TRPT_j$	:	Total remaining processing time of $A_j, j \in A$
SLK	:	Slack for activity $A_j, j \in A$
ω_j	:	Weight on the link from $A_j$to resource layer
α	:	Learning coefficient
ε_k	:	Error in iteration k
I	:	Initial dummy activity node
F	:	Final dummy activity node

τ_j	:	Threshold value of A_j = Number of tasks immediately preceding A_j, $j \in A \cup F$
t	:	Elapsed time in current iteration
ST_j	:	Start time of activity j.
PT_j	:	Processing time of activity j
LST_j	:	Latest start time of activity j
LFT_j	:	Latest finish time of activity j
PR_j	:	Set of tasks that immediately precede task j, $j \in A \cup F$
SU_j	:	Set of tasks immediately succeeding task j, $j \in A$
Win_j	:	Winning status of A_j, $j \in A$
SCP	:	Set of tasks currently in process.

Following are all functions of elapsed time t:

$IA_j(t)$	:	Input function value of activity node j, $j \in I \cup A \cup F$
$IRLA_j(t)$	:	Input function value of resource layer from activity node A_j, $j \in A$
$IRLRL_{jk}(t)$	:	Input function value of resource layer j from other resource layers for each resource type k, $j \in A$, $k \in R$
$OA_j(t)$	:	Output function value of activity node j, $j \in I \cup A \cup F$
$ORLF_{jp}(t)$	:	Output of RL_j to activity node A_p in the forward direction, $j \in A$, $p \in SU_j$
$ORLR_j(t)$	:	Output of Resource layer RL_j to activity node A_j in reverse direction, $j \in A$
$ORLL_{jp}(t)$	:	Output of Resource layer RL_j to RL_p in lateral direction, $j,p \in A$, $j \neq p$
$\theta A_j(t)$	:	Activation function of activity node j, $j \in A$
$\theta RL_j(t)$	:	Activation function of Resource layer RL_j, $j \in T$
$assign_j(t)$	:	Activity j assigned at time t
$S(t)$	:	Set of activities that can start at time t. $S(t) = \{A_j \mid OA_j(t) = 1\}$
$RAv_k(t)$	:	Number of resources of type k available at time t

12.4.2 Preliminary Steps

1 Calculate the Lower Bound, which is the same as the critical path duration under infinite resource availability assumption.

2 Weights (ω_j) are initialized at 10.00. The value of 10 was arrived at after some computational experience. The value of the initial weights should be such that after subsequent modification to weights, the value should remain positive. The choice of the value of initial weight therefore also depends on the value of the learning coefficient used.

3 Calculate the threshold of each task τ_j. The threshold of activity j is defined as the number of tasks immediately preceding task j. The threshold value is used to determine when a task is ready to start.

12.4.3 AugNN Functions

The neural network algorithm can be described with the help of the learning strategy and the input functions, the activation functions and the output functions for the task nodes and the machine nodes.

12.4.3.1 Activity layer functions. Input functions, activation states and output functions are now explained for the nodes on the activity layer.

Input function

$\forall j \in A \cup F \quad IA_j(0) = 0$

$\forall j \in I$ *(the starting signal of the initial dummy node is 1)* $\quad IA_j(0) = 1$

$\forall q \in PR_j, j \in T \cup F \quad IA_j(t) = IA_j(t-1) + \sum_q ORLF_{qj}(t)$

IA_j helps to enforce precedence constraint. When IA_j becomes equal to τ_j, the activity can be assigned.

Activation function

Activity nodes' initial activation state (i.e. at t=0) is 1. $\forall j \in T$

$$\theta A_j(t) = \begin{cases} 1 & \text{if} \quad IA_j(t) < \tau_j \\ 2 & \text{if} \quad (\theta A_j(t-1) = 1 \vee 2) \wedge IA_j(t) = \tau_j \\ 3 & \text{if} \quad (\theta A_j(t-1) = 2 \vee 3) \wedge ORLR_j(t) < 0 \\ 4 & \text{if} \quad (\theta A_j(t-1) = 4 \vee (\theta A_j(t-1) = 3 \wedge ORLR_j(t) = 0) \end{cases}$$

Note: For the initial dummy node, $\tau_j = 1$

State 1 above implies that activity j is not ready to be assigned because input to activity j is less than its threshold τ. State 2 implies that activity j is ready to be assigned because its input equals its threshold. State 3 implies that the activity is in process because it is receiving a negative signal from the resource layer that it is currently being processed. State 4 implies that the activity is complete and the negative signal from the resource layer is no longer there.

Output function

$$OA_j(t) = \begin{cases} 1 \text{ if } \theta A_j(t) = 2 \\ 0 \text{ otherwise} \end{cases}$$

If an activity is ready to start but not assigned yet, it sends a unit signal to the resource layer.

F-Node

$$OA_j(t) = \begin{cases} t \text{ if } IA_F(t) = \tau_j \\ 0 \text{ otherwise} \end{cases}$$

The final node outputs the makespan t, the moment its threshold point is reached.

12.4.3.2 Resource layer functions. Input, activation and output functions of resource layers are now explained.

Input function

$\forall j \in A \quad IRLA_j(t) = OA_j(t) * \omega_j$

This is the weighted output from activity node j. Whenever it is positive, it means that the resources are being requested by activity j for assignment.

$\forall j \in A, k \in R \quad IRLRL_{jk}(t) = \sum_{p \in SCP} ORLL_{pjk}$

Activation function

Let $\chi_j(t) = IRL_j(t) * TaskHeuristicParameter_j$

Let *RAv$_j$(t) : Whether resources for activity j are available or not*

$$RAv_j(t) = \begin{cases} 1 & \text{if} \quad \forall k \in R \ (b_k - IRLRL_{jk} \geq r_{jk}) \\ 0 & \text{otherwise} \end{cases}$$

$$assign_j(t) = \begin{cases} 1 & \text{if} \quad RAv_j(t) = 1 \wedge \\ & \chi_j(t) = \max[\chi_j(t) | A_j \in S(t)] \wedge \forall j \in A \ \chi_j(t) > 0 \\ 0 & otherwise \end{cases}$$

The assignment takes place if the product of Input of the resource layer and the Heuristic dependent activity parameter is positive and highest and if the resources are available. The requirement for highest is what enforces the chosen heuristic.

TaskHeuristicParameter is a task parameter dependent on the chosen heuristic.

$$TaskHeuristicParameter = \begin{cases} TRPT & \text{for } TRPT \text{ heuristic} \\ LFT & \text{for } LFT \text{ heuristic} \\ EST & \text{for } EST \text{ heuristic} \\ EFT & \text{for } EFT \text{ heuristic} \\ LST & \text{for } LST \text{ heuristic} \\ RND & \text{for } RANDOM \text{ heuristic} \end{cases}$$

If $assign_j(t) = 1$, then $ST_j = t$.

If $|S(t)| > 1$ then $Win_j = 1$.

Resource layers' Initial Activation State (i.e. at $t = 0$) is 1. $\forall i \in M, j \in T$,

$$\theta RL_j(t) = \begin{cases} 1 & \text{if} \quad RAv_j(t) = 1 \\ 2 & \text{if} \quad \theta RL_j(t-1) = 1 \vee \theta RL_j(t) = 1) \wedge assign_j(t) = 1 \\ 3 & \text{if} \quad (\theta RL_j(t-1) = 2 \vee 3) \wedge t < ST_j + PT_j : \\ 4 & \text{if} \quad \theta RL_j(t-1) = 3 \wedge t = ST_j + PT_j \end{cases}$$

State 1 implies that the resources are available. State 2 implies that the resources are busy and that they were just assigned. State 3 implies that the resources are busy and state 4 implies that the resources were just released.

Output function

$$ORLF_{jp}(t) = \begin{cases} 1 & \text{if } \theta RL_j(t) = 4 \\ 0 & \text{if } \theta RL_j(t) = 1,2,3 \end{cases} \quad j \in A, p \in SU_j$$

$$ORLR_j(t) = \begin{cases} -1 & \text{if } \theta RL_j(t) = 2,3 \\ 0 & \text{if } \theta RL_j(t) = 1,4 \end{cases} \quad j \in A$$

$$ORLL_{pjk}(t) = \begin{cases} r_{jk} & \text{if } \theta RL_j(t) = 2,3 \\ 0 & \text{if } \theta RL_j(t) = 1,4 \end{cases} \quad j,p \in A, k \in R, p \neq j$$

The output of F represents the makespan and the $assign_j(t)$ gives the schedule. If a resource is either assigned or released during a certain time unit, all functions need to be recalculated without incrementing the time period.

12.4.4 Learning Strategy

A learning strategy is required to modify the weights. In this study we used the following learning strategy:

Winning activities:

If $\forall j \in A \quad Win_j = 1$ then $(\omega_j)_{k+1} = (\omega_j)_k - \alpha * TaskParameter_j * \varepsilon_k$

Non-winning activities:

If $\forall j \in A \quad Win_j = 0$ then $(\omega_j)_{k+1} = (\omega_j)_k + \alpha$ *$TaskParameter_j * \varepsilon_k$

In addition to these weight changes in each iteration, we propose two additional features that govern learning, namely, reinforcement and backtracking. These features are explained here briefly.

Reinforcement:

Neural Networks use the concept of positive reinforcement of weights if the network performs well. We implement this idea of reinforcement by implementing the following rule. If in a particular iteration the makespan improves, the weight changes of that iteration with respect to the previous iteration are magnified by a factor called the reinforcement factor (RF).

$$(\omega_{ik})_{k+1} = (\omega_{ik})_{k+1+RF*}((\omega_{ik})_{k+1} - (\omega_{ik})_k)$$

Backtracking:
Sometimes it is possible to not obtain any improvement over several iterations. When this happens, it is best to abandon that search path and start over from the previous best solution weights. We can parameterize how many iterations of no improvement to tolerate. This backtracking technique was part of our learning strategy. In order to do this, we store the set of weights corresponding to the best solution obtained so far and revert back to it whenever solution does not improve for some iterations.

12.4.5 End of iteration routines

1 Calculate the gap which is the difference between obtained makespan and the lower bound

2 Store the best solution so far.

3 If the lower bound is reached, stop the program.

4 If the number of iterations exceeds a certain specified number, such as 1000 or 5000, stop the program.

5 If continuing with the next iteration, modify weights using the learning strategy. Apply backtracking and reinforcement, whenever necessary.

12.5 Computational experiments and results

We now present the result of the computational tests and compare them with the best published algorithms. The HNA approach algorithms were coded in Visual Basic 6.0 and run on a Celeron 2300 MHz personal computer. Well known benchmark problem instance sets (Kolisch et al (1995) and Kolisch and Sprecher (1996)) are used to evaluate the algorithm (PSPLIB, http://www.bwl.uni-kiel.de /Prod/psplib/ index.html). The sets J30 and J60 consists of 480 problem instances with four resource types and 30 and 60 activities, respectively. The set J120 consists of 600 problem instances with four resource type and 120 activities.

In our implementation, the learning coefficient α is set to 0.001 and the weights are initialized at 10. An initial solution is generated using a priority rules such as LFT or EST. The weights are modified after each iteration, using the learning strategy. The stopping criterion is to stop if the solution is equal to the lower bound or if a predetermined number of maximum schedules is reached. We set the maximum number of schedules to 1000 and 5000.

Tables 16.1 through 16.3 display the results obtained by our algorithm and other tested heuristics for 1,000 and 5,000 schedules, respectively. In these

Table 12.1. Percent Average Deviations from the Optimum Solution: Comparative Results for the J30 Problems

			# of Schedules	
Algorithm	**SGS**	**Reference**	**1,000**	**5,000**
Sampling - LFT – FBI	both	Tormos and Lova (2003)	0.23	0.14
GA - FBI	both	Alcaraz et al (2004)	0.25	0.06
HNA - FBI	**both**	**this paper**	**0.25**	**0.11**
Sampling - LFT - FBI	both	Tormos and Lova (2001)	0.25	0.15
GA – hybrid, FBI	serial	Valls et al (2003)	0.27	0.06
Scatter Search – FBI	serial	Debels et al (2004)	0.27	0.11
GA – FBI	serial	Alcaraz and Maroto (2001)	0.33	0.12
GA – FBI	serial	Valls et al (2005)	0.34	0.20
GA - Self adapting	both	Hartmann (2002)	0.38	0.22
SA - Activity List	serial	Bouleimen and Lecocq (2003)	0.38	0.23
TS - Activity List	serial	Nonobe and Ibaraki (2002)	0.46	0.16
Sampling – FBI	serial	Valls et al (2005)	0.46	0.28
GA - Activity List	serial	Hartmann (1998)	0.54	0.25
Adaptive Sampling	both	Schirmer (2000)	0.65	0.44
GA	serial	Coelho and Tavares (2003)	0.74	0.33
Adaptive Sampling	both	Kolisch and Drexl (1996)	0.74	0.52
Sampling – Global	serial	Coelho and Tavares (2003)	0.81	0.54
TS		Baar et al (1997)	0.86	0.44
GA – Random Key	serial	Hartmann (1998)	1.03	0.56
GA – Priority Rule	serial	Hartmann (1998)	1.38	1.12
Sampling – WCS	parallel	Kolisch (1996b)	1.40	1.28
Sampling - LFT	parallel	Kolisch (1996b)	1.40	1.29
Sampling – random	serial	Kolisch (1995)	1.44	1.00
Sampling – random	parallel	Kolisch (1995)	1.77	1.48
GA		Leon and Ramamoorthy (1995)	2.08	1.59

tables we present the type of heuristics, the type of schedule generation scheme used, the authors of each heuristic and the average deviation from the critical path based lower bound (from the optimal solution for J30 instances) for 1000 and 5000 schedules, respectively. In each table, the heuristics are sorted according to descending performance with respect to 1000 schedules.

Table 16.1 presents the percentage deviations from the optimal makespan for the instance set J30 in which all problem instances have been solved to optimality by Demeulemeester and Herroelen (1997) branch and bound procedure. Our algorithm solved 448 out of 480 problems to optimality and the

average deviation from the optimal solution is just 0.25 and 0.11 percent for 1000 schedules and 5000 schedules, respectively.

Table 12.2. Percent Average Deviations from the Critical-Path-Based Lower Bound: Comparative Results for the J60 Problems

			# of Schedules	
Algorithm	**SGS**	**Reference**	**1,000**	**5,000**
GA – hybrid, FBI	serial	Valls et al (2003)	11.56	11.10
HNA - FBI	**both**	**this paper**	**11.72**	**11.39**
Scatter Search – FBI	serial	Debels et al (2004)	11.73	11.10
GA - FBI	both	Alcaraz et al (2004)	11.89	11.19
Sampling - LFT – FBI	both	Tormos and Lova (2003)	12.04	11.72
Sampling - LFT - FBI	both	Tormos and Lova (2001)	12.11	11.82
GA – FBI	serial	Valls et al (2005)	12.21	11.27
GA - Self adapting	both	Hartmann (2002)	12.21	11.70
GA – FBI	serial	Alcaraz and Maroto (2001)	12.57	11.86
GA - Activity List	serial	Hartmann (1998)	12.68	11.89
Sampling – FBI	serial	Valls et al (2005)	12.73	12.35
SA - Activity List	serial	Bouleimen and Lecocq (2003)	12.75	11.90
Adaptive Sampling	both	Schirmer00	12.94	12.59
TS - Activity List	serial	Nonobe and Ibaraki (2002)	12.97	12.18
GA	serial	Coelho and Tavares (2003)	13.28	12.63
GA – Priority Rule	serial	Hartmann (1998)	13.30	12.74
Adaptive Sampling	both	Kolisch and Drexl (1996)	13.51	13.06
Sampling - LFT	parallel	Kolisch (1996b)	13.59	13.23
Sampling – WCS	parallel	Kolisch (1996b)	13.66	13.21
Sampling – Global	serial	Coelho and Tavares (2003)	13.80	13.31
TS		Baar et al (1997)	13.80	13.48
GA		Leon and Ramamoorthy (1995)	14.33	13.49
GA – Random Key	serial	Hartmann (1998)	14.68	13.32
Sampling – random	parallel	Kolisch (1995)	14.89	14.30
Sampling – random	serial	Kolisch (1995)	15.94	15.17

For J60 problems set, some of the optimal solutions are not known, so we measure the average percentage deviation from the critical-path based lower bound for comparison purposes. Table 16.2 summarizes the results for J60 test instances. 295 out of 480 instances are solved to critical path based lower bound. The average deviation from the critical path based lower bound is 11.72 and 11.39 percent for 1000 and 5000 schedules, respectively.

Table 12.3. Percent Average Deviations from the Critical-Path-Based Lower Bound: Comparative Results for the J120 Problems

			# of Schedules	
Algorithm	**SGS**	**Reference**	**1,000**	**5,000**
GA – hybrid, FBI	serial	Valls et al (2003)	34.07	32.54
HNA - FBI	**both**	**this paper**	**34.94**	**34.57**
Scatter Search – FBI	serial	Debels et al (2004)	35.39	33.24
GA – FBI	serial	Valls et al (2005)	35.98	35.30
Sampling - LFT – FBI	both	Tormos and Lova (2003)	35.98	35.30
Sampling - LFT - FBI	both	Tormos and Lova (2001)	36.32	35.30
GA - FBI	both	Alcaraz et al (2004)	36.53	33.91
GA - Self adapting	both	Hartmann (2002)	37.19	35.39
Sampling – FBI	serial	Valls et al (2005)	38.21	37.47
GA – FBI	serial	Alcaraz and Maroto (2001)	39.36	36.57
GA - Activity List	serial	Hartmann (1998)	39.37	36.74
Sampling - LFT	parallel	Kolisch (1996b)	39.60	38.75
Sampling – WCS	parallel	Kolisch (1996b)	39.65	38.77
Adaptive Sampling	both	Schirmer (2000)	39.85	38.70
GA – Priority Rule	serial	Hartmann (1998)	39.93	38.49
GA	serial	Coelho and Tavares (2003)	39.97	38.41
TS - Activity List	serial	Nonobe and Ibaraki (2002)	40.86	37.88
Sampling – Global	serial	Coelho and Tavares (2003)	41.36	40.46
Adaptive Sampling	both	Kolisch and Drexl (1996)	41.37	40.45
SA - Activity List	serial	Bouleimen and Lecocq (2003)	42.81	37.68
Sampling – LFT	Serial	Kolisch (1996b)	42.84	41.84
GA		Leon and Ramamoorthy (1995)	42.91	40.69
Sampling – random	parallel	Kolisch (1995)	44.36	43.05
GA – Random Key	serial	Hartmann (1998)	45.82	45.25
Sampling – random	serial	Kolisch (1995)	49.25	47.61

Table 16.3 summarizes the results for J120 set. 155 out of 600 problems matched the critical path based lower bound. The average deviations are 34.94 and 34.57 percent, respectively, for 1000 and 5000 schedules.

12.6 Conclusions

We proposed, developed and tested a new metaheuristic approach based on the principles of neural networks. Augmented-neural-network approach was used for parallel schedule generation and adaptive-learning approach for

serial schedule generation scheme. We called this approach the hybrid neural approach (HNA). To the best of our knowledge, this is the first time that neural-networks based metaheuristics have been applied to the RCPSP. So, research in this approach is still in its infancy. We tested this approach on some well-known RCPSP benchmark problem instances in the literature. The computational results are very encouraging as they compare very well with some of the best results in the literature from techniques such as tabu search, simulated annealing, genetic algorithms and scatter search. The approach, in spite of being relatively new, gave very good results, and therefore appears to be very promising and worthy of further exploration.

Future research may focus on developing some hybrid approaches involving the HNA approach and some of the other successful approaches such as genetic algorithms and scatter search, to further improve the results. This new approach should also be applied to multi-mode resource constrained project scheduling problems with renewable and non-renewable resources.

References

Agarwal, A., Jacob, V.S. and Pirkul, H. (2003). Augmented neural networks for task scheduling, *European Journal of Operational Research*, 151(3):481–502.

Agarwal, A., Colak, S. and Eryarsoy, E. (2005). Improvement heuristic for the flow-shop scheduling problem: an adaptive-learning approach, *European Journal of Operational Research*, 169(3):801–815.

Alcaraz, J. and Maroto, C. (2001). A robust genetic algorithm for resource allocation in Project Scheduling, *Annals of Operations Research*. 102:83–109.

Alcaraz, J., Maroto, C. and Ruiz, R. (2004). Improving the performance of genetic algorithms for the RCPS problem, *Proceedings of the Ninth International Workshop on Project Management and Scheduling*, pp. 40–43.

Alvares-Valdes, R. and Tamarit, J.M. (1989). Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis, in: *Advances in Project Scheduling*, R. Slowinski, J. Weglarz (Eds.), Elsevier, Amsterdam, pp. 113–134.

Baar, T., Brucker, P. and Knust, S. (1997). Tabu search algorithms for resource-constrained project scheduling problems, in: *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimisation*, S. Voss, S. Martello, I. Osman, C. Roucairol (Eds.), Kluwer, pp. 1–18.

Blazewicz, J., Lenstra, J.K. and Rinnooy Kan A.H.G. (1983). Scheduling projects to resource constraints: classification and complexity, *Discrete Applied Mathematics*. 5:11–24.

Boctor, F.F. (1990). Some efficient multi-heuristic procedures for resource-constrained project scheduling, *European Journal of Operational Research* 49:3–13.

Boctor, F.F. (1996). An adaptation of the simulated annealing algorithm for solving resource-constrained project scheduling problems, *International Journal of Production Research.* 34:2335–2351.

Bouleimen, K. and Lecocq, H. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version, *European Journal of Operational Research.* 149:268–281.

Brucker, P., Knust, S., Schoo, A. and Thiele, O. (1998). A branch & bound algorithm for the resource-constrained project scheduling problem, *European Journal of Operational Research* 107(2):272–288.

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

Christofides, N., Alvarez-Valdes, R. and Tamarit, J.M. (1987). Project scheduling with resource constraints: a branch-and-bound approach, *European Journal of Operational Research* 29(2):262–273.

Cho, J.H. and Kim, Y.D. (1997). A simulated annealing algorithm for resource-constrained project scheduling problems, *Journal of the Operational Research Society.* 48:736–744.

Coelho, J. and Tavares, L. (2003). Comparative analysis of meta–heuricstics for the resource constrained project scheduling problem, Technical report, Department of Civil Engineering, Instituto Superior Tecnico, Portugal.

Cooper, D.F. (1976). Heuristics for scheduling resource–constrained projects: An experimental investigation, *Management Science* 22:1186–1194.

Davis, E.W. and Heidorn, G.E. (1971). An algorithm for optimal project scheduling under multiple resource constraints, *Management Science* 17:803–816.

Davis, E.W. and Patterson, J.H. (1975). A comparison of heuristic and optimum solutions in resource-constrained project scheduling, *Management Science.* 21:944–955.

Debels, D., De Reyck, B., Leus, R. and Vanhoucke, M. (2006). A hybrid scatter search/electromagnetism meta-heuristic for project scheduling, *European Journal of Operational Research* 169(2):638–653.

Demeulemeester, E. and Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problems, *Management Science.* 38(12):1803–1818.

Demeulemeester, E. and Herroelen, W. (1997). New benchmark results for the resource-constrained project scheduling problem, *Management Science* 43(11):1485–92.

Dorndorf, U., Pesch, E. and Phan-Huy, T. (2000). A branch-and-bound algorithm for the resource-constrained project scheduling problem, *Mathematical Methods of Operations Research*, 52:413-439.

Drexl, A. (1991). Scheduling of project networks by job assignment, *Management Science*. 37:1590-1602.

Foo, Y.P.S. and Takefuji, Y. (1988). Stochastic neural networks for solving job-shop scheduling, *Proceedings of Joint International Conference on Neural Networks Vol. 2*, pp. 275-290.

Hartmann, S. (1998). A competitive genetic algorithm for the resource-constrained project scheduling, *Naval Research Logistics*, 45:733–750.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints, *Naval Research Logistics*. 49:433–448.

Hartmann, S. and Kolisch, R. (2000). Experimental evaluation of state–of–the–art heuristics for the resource–constrained project scheduling problem, *European Journal of Operational Research* 127:394–407.

Herroelen, W., Demeulemeester, E. and De Reyck, B. (1998). Resource- constrained project scheduling: A survey of recent developments, *Computers & Operations Research* 25(4):279-302.

Hindi, K. S., Yang H. and Fleszar, K. (2002). An evolutionary algorithm for resource-constrained project scheduling, *IEEE Transactions on Evolutionary Computation*. 6:512–518.

Hopfield, J.J. and Tank, D.W. (1985). Neural computation of decisions in optimization problems, *Biological Cybernetics*. 52:141-152.

Icmeli, O., Erenguc, S.S. and Zappe, C.J. (1993). Project scheduling problems: A survey, *International Journal of Operations & Production Management* 13(11):80-91.

Kolisch, R. (1995). *Project scheduling under resource constraints: efficient heuristics for several problem classes*, Physica, Heidelberg, Germany.

Kolisch, R. (1996a). Efficient priority rule for the resource-constrained project scheduling problem, *Journal of Operations Management*. 14(3):179–192.

Kolisch, R. (1996b). Serial and parallel resource–constrained project scheduling methods revisited: theory and computation, European *Journal of Operational Research* 90:320–333.

Kolisch, R. and Drexl, A. (1996). Adaptive search for solving hard project scheduling problems, *Naval Research Logistics*. 43:23–40.

Kolisch, R. and Hartmann, S. (2005). Experimental investigation of heuristics for resource–constrained project scheduling: An update, *European Journal of Operational Research*, forthcoming.

Kolisch, R. and Padman, R. (2001). An integrated survey of deterministic project scheduling, OMEGA. 29:249–272.

Kolisch R. and Sprecher, A. (1996). PSPLIB - A project scheduling problem library. *European Journal of Operational Research*, 96(1):205–216.

Kolisch, R., Sprecher, A. and Drexl, A. (1995). Characterisation and generation of a general class of resource-constrained project scheduling problem, *Management Science*, 41(10): 1693–1703.

Lee, J.K. and Kim, Y.D., 1996, Search heuristics for resource-constrained project scheduling, *Journal of the Operational Research Society*. 47:678–689.

Leon, V.J. and Ramamoorthy, B. (1995). Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling, *OR Spektrum*. 17:173–182.

Mingozzi, A. , Maniezzo, V., Ricciardelli, S. and Bianco, L. (1998). An exact algorithm for project scheduling with resource constraints based on new mathematical formulation, *Management Science,* 44(5):714-29.

Merkle, D., Middendorf, M. and Schmeck H. (2002). Ant colony optimization for resource-constrained project scheduling, *IEEE Transactions on Evolutionary Computation*. 6:333–346.

Nonobe, K. and Ibaraki, T. (2002). Formulation and tabu search algorithm for the resource constrained project scheduling problem, in: *Essays and Surveys in Metaheuristics*, C. C. Ribeiro and P. Hansen, eds, Kluwer Academic Publishers, pp. 557–588.

Ozdamar, L. and Ulusoy, G., (1994). A local constraint based analysis approach to project scheduling under general resource constraints, *European Journal of Operational Research*. 79:287–298.

Ozdamar, L. and Ulusoy, G. (1995). A survey on the resource-constrained project scheduling problem, *IIE Transactions*. 27:574-586.

Patterson, J.H. and Huber, W.D. (1974). A horizon-varying, zero-one approach to project scheduling, *Management Science* 20:990–998.

Patterson, J.H. and Roth, G.W., 1976, Scheduling a project under multiple resource constraints: a zero-one programming approach, *AIIE Transactions*. 8:449–55.

Pinson, E., Prins, C. and Rullier, F. (1994). Using tabu search for solving the resource-constrained project scheduling problem, in: *Proceedings of the 4th International Workshop on Project Management and Scheduling*, Leuven, Belgium, pp. 102–106.

Pritsker, A.A.B., Watters, L.J. and Wolfe, P.M. (1969). Multiproject scheduling with limited resources: a zero-one programming approach, *Management Science* 16: 93–107.

Sabuncuoglu, I. and Gurgun, B. (1996). A neural network model for scheduling problems, *European Journal of Operational Research* 93:288-299.

Schirmer, A. and Riesenberg, S. (1998). Class-based control schemes for parameterized project scheduling heuristics, Manuskripte aus den Instituten für Betriebswirtschaftslehre 471, Universität Kiel, Germany.

Schirmer, A. (2000). Case–based reasoning and improved adaptive search for project scheduling, *Naval Research Logistics*.47:201–222.

Talbot, F.B. and Patterson, J.H. (1978). An efficient integer programming algorithm with network cuts for solving resource constrained scheduling problems, *Management Science*, 24(11):1163–74.

Thomas, P. R. and Salhi, S. (1998). A tabu search approach for the resource constrained project scheduling problem, *Journal of Heuristics*, 4:123–139.

Toklu, Y.C. (2002). Application of genetic algorithms to construction scheduling with or without resource constraints, *Canadian Journal of Civil Engineering*. 29:421-429.

Tormos, P. and Lova, A., 2001, A competitive heuristic solution technique for resource constrained project scheduling, *Annals of Operations Research*. 102:65–81.

Tormos, P. and Lova, A. (2003). An efficient multi-pass heuristic for project scheduling with constrained resources, *International Journal of Production Research*. 41(5):1071–1086.

Ulusoy, G. and Ozdamar, L. (1989). Heuristic performance and network/ resource characteristics in resource-constrained project scheduling, *Journal of the Operational Research Society*. 40:1145–1152.

Valls, V., Ballestin, F. and Quintanilla, M. S. (2003). A hybrid genetic algorithm for the RCPSP, Technical report, Department of Statistics and Operations Research, University of Valencia.

Valls, V., Ballestin, F. and Quintanilla M. S. (2005). Justification and RCPSP: A technique that pays, *European Journal of Operational Research*. 165(2):375-386.