periodica polytechnica

Electrical Engineering and Computer Science
57/2 (2013) 57–64
doi: 10.3311/PPee.2090
http://periodicapolytechnica.org/ee

RESEARCH ARTICLE

Improvements to the Hopfield Neural Network Solution of the TWT Scheduling Problem

Kálmán Tornai / *Norbert* Fogarasi / *János* Levendovszky

Received 2013-04-10, revised 2013-06-02, accepted 2013-06-03

Abstract

This paper explores novel, polynomial time, heuristic, approximate solutions to the NP-hard problem of finding the optimal job schedule on identical machines which minimizes total weighted tardiness (TWT). We map the TWT problem to quadratic optimization and demonstrate that the Hopfield Neural Network (HNN) can successfully solve it. Furthermore, the solution can be significantly sped up by choosing the initial state of the HNN as the result of a known simple heuristic, we call this Smart Hopfield Neural Network (SHNN). We also demonstrate, through extensive simulations, that by considering random perturbations to the Largest Weighted Process First (LWPF) and SHNN methods, we can introduce further improvements to the quality of the solution, we call the latter Perturbed Smart Hopfield Neural Network (PSHNN). Finally, we argue that due to parallelization techniques, such as the use of GPGPU, the additional cost of these improvements is small. Numerical simulations demonstrate that PSHNN outperforms HNN in over 99% of all randomly generated cases by an average of 3-7%, depending on the problem size. On a specific, large scale scheduling problem arising in computational finance at Morgan Stanley, one of the largest financial institutions in the world, PSHNN produced a 5% improvement over the next best heuristic.

Keywords

Scheduling · Optimization · Quadratic programming · Neural networks

Kálmán Tornai

Pázmány Péter Catholic University, Faculty of Information Technology, Práter u. 50/a., H-1083 Budapest, Hungary
e-mail: tornai.kalman@itk.ppke.hu

Norbert Fogarasi

Department of Networked Systems and Services, Budapest University of Technology and Economics, Magyar tudósok krt. 2., H-1117 Budapest, Hungary

Managing Director, Morgan Stanley Hungary Analytics, Lechner Ödön fasor 8., H-1095 Budapest, Hungary

János Levendovszky

Department of Networked Systems and Services, Budapest University of Technology and Economics, Magyar tudósok krt. 2., H-1117 Budapest, Hungary

1 Introduction

The recent increase of interest in classical problems of scheduling theory can be attributed to the improvements in computer hardware over the last two decades, which have made applications of distributed computing possible to a variety of new disciplines, including telecommunication, computational biology, chemistry and finance (Brucker [2], Pinedo [13]). Running large scale Monte-Carlo simulations in real-time, requires thoughtful distribution of tasks on identical machines, so scheduling receives a central interest in financial computing. In our earlier work (Fogarasi et al [5]), we outlined some specific current applications in computational finance as motivation for examining the problem of scheduling jobs with predefined *deadlines* and *weights*, allowing *pre-emption* (the stopping and resumption of work items), on *identical* machines in a way that the *total weighted tardiness* (TWT) of all jobs is minimal. In that paper, we gave a detailed review of previous work on this problem, dating back to the 1970's. An important result is the proof by Du and Leung [3] that even the single machine, total tardiness problem is NP-hard. This finding has directed many researchers to take the approach of finding approximate solutions to the problem in polynomial time. Akyol and Bayhan [1] provide an excellent recent review of artificial neural network based approaches to scheduling problems and proposes a coupled gradient network to solve the weighted earliness plus tardiness problem on multiple machines. The feasibility of the method is illustrated on a single 8-job scheduling problem, but there is no mention about the scalability of the method to larger problem sizes which is the focus of our study. Although there have been many other attempts (e.g. Maheswaran [11], there is no traceable work on empirically demonstrated reliable polynomial time heuristic which scales to over 100-job scheduling problems and outperforms the existing simple heuristics. To fill this gap, we suggested a novel approach based on the HNN approach which is based on mapping the problem into quadratic optimization and we showed empirically that it reliably outperforms the Earliest Due Date (EDD), Weighted Shortest Processing Time (WSPT) and our own Least Weighted Process First (LWPF) heuristics.

In this paper, we take the HNN approach as our starting point, and we examine improvements by intelligent selection of the initial state for the algorithm (SHNN) and considering perturbations to this initial state (PSHNN). We compare this to the approach of simply perturbing the results of simpler heuristics (PLWPF) and we also study the runtime characteristics of these methods, taking into consideration parallelization possibilities offered by technologies such as GPGPU. In order to demonstrate the practical viability of our methods, we examine a large-scale scheduling problem arising at Morgan Stanley with the overnight batch of interest rate derivatives hedge computations. This is a critically important practical application, as investment banks spend millions of dollars each year on hardware to perform these daily calculations which are required by the trading desks, senior risk management and most recently by external regulators. We demonstrate that the improvements provided by PSHNN are significant when compared to other heuristic methods, and thus has the potential of decreasing hardware spend by up to 5% by the financial services companies.

The paper is organized as follows: in Section 2, we give a formal definition of the problem; in Section 3, we review existing heuristic methods; in Section 4 the novel heuristics are introduced; in Section 5, numerical results are presented including a practical case study on a large problem size; and finally in Section 6, some conclusions are drawn and directions for future research are outlined.

2 Problem Formulation

In this section, we give a formal presentation for the problem of optimally scheduling jobs on identical processors with the purpose of minimizing total weighted tardiness relative to a prescribed job deadline schedule.

Given N jobs with sizes

$$\mathbf{x} = \{x_1, x_2, x_3, \ldots, x_N\} \in \mathcal{N}^N,$$

we assume that the processing of the jobs can be stopped and resumed at any time on any machine. In the literature this condition is known as preemption (Brucker [2]). An ordered set of cut-off times:

$$\mathbf{K} = \{K_1, K_2, K_3, \ldots, K_N\} \in \mathcal{N}^N$$

is also given, which defines the time within which the jobs are to be completed. The constant number of identical machines available for scheduling is denoted by $V \in \mathcal{N}$. Vector

$$\mathbf{w} = \{w_1, w_2, w_3, \ldots, w_N\} \in \mathcal{R}^N, w_i > 0, \forall i = 1, \ldots, N$$

denotes the relative priority (or weight) of each job. A schedule is represented by a binary matrix $\mathbf{C} \in \{0,1\}^{N \times L}$ where $C_{i,j} = 1$ if job i is being processed at time slot j, and L denotes the length of the schedule. An example is given in (1), where the parameters are the following:

$$V = 2, N = 3, \mathbf{x} = \{2,3,1\}, \mathbf{K} = \{3,3,3\}$$

$$\mathbf{C} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{1}$$

The first row in (1) denotes the fact that under this schedule **C**, the first job is processed in time steps 1 and 3 (note that preemption is used as the processing of this job is not continuous) and therefore the processing unit size 2 of this job completes within the prescribed cut-off time of time step 3. Similarly, the 3 units of the second job complete within the cut-off time of 3 and the third job is completed ahead of the cut-off time on time step 2. Summing the columns of matrix **C**, we see that the maximal capacity of $V = 2$ is fully utilized on each time step.

In order to evaluate the effectiveness of a given schedule **C**, we define *tardiness* of a job as follows (Naderi et al. [12]):

$$T_i = \max(0, F_i - K_i) \tag{2}$$

where F_i is the actual finish time of job i under schedule **C** : $F_i = \text{argmax}_j \left\{C_{i,j} = 1\right\}$, the position of the last 1 in the i^{th} row in scheduling matrix **C** .

The problem to be solved can now be stated formally as finding the optimal schedule for which

$$\mathbf{C}_{opt} := \underset{c}{\text{argmin}} \sum_{i=1}^{N} w_i T_i \tag{3}$$

under the following two constraints:

1 The sum of each row in the scheduling matrix is equal to the given job sizes, i.e.

$$\sum_{j=1}^{L} C_{i,j} = x_i, \forall i = 1, \ldots, N \tag{4}$$

2 The number of scheduled jobs at any given time instant does not exceed the capacity of the system:

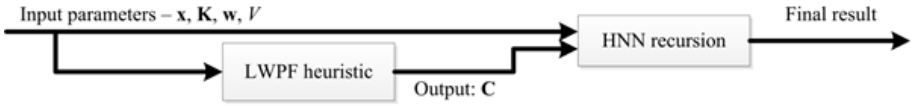
$$\sum_{i=1}^{L} C_{i,j} \leq V, \forall i = 1, \ldots, N \tag{5}$$

We now revisit our previous example with a minor change: $V = 2, N = 3, \mathbf{x} = \{2,3,2\}, \mathbf{K} = \{3,3,3\}$ and a weight vector $\mathbf{w} = \{3,2,1\}$. It can be observed that there is no solution, in which all jobs are completed before their cut-off times. A minimal weighted tardiness solution is the following:

$$\mathbf{C} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \tag{6}$$

In this case, the first two jobs are as in the previous example, but job 3 could not be completed by the cut-off time of 3 and because one unit of work is finished after the cut-off time and the weight associated with this job is 1, the schedule has a TWT measure of 1.

Fig. 1. Block diagram of the SHNN method



3 Existing Heuristic Methods

Given the NP-hard nature of the scheduling task as proven by Du and Leung [3], to find the exact, optimal solution, in most real-world settings is not feasible. Thus we follow a pragmatic approach of finding a fast, sub-optimal, but good solution. We outline the recently developed Hopfield Neural Network (HNN) based solution as well as the Largest Weighted Process First (LWPF) heuristic.

3.1 Largest Weighted Process First (LWPF) Heuristic

This heuristic re-labels the sequence of jobs to be executed in non-decreasing order of weights. Using the notation of Section 2, we re-label the job indices so that following inequality holds:

$$w_1 \geq w_2 \geq \ldots \geq w_n. \tag{7}$$

Once this re-indexing is completed, the jobs are allocated to the machines in this order, always utilizing the maximum available capacity. When a job finishes and capacity is thus freed up for the next time step, the next job in the queue is scheduled. This heuristic has been empirically proven to have very good relative performance, compared to other simple heuristics such as the Earliest Due Date (EDD) and Weighted Shortest Processing Time (WSPT) methods (Fogarasi et al. [5]).

3.2 Hopfield Neural Network (HNN) based solution

This approach transforms the original constrained optimization into a quadratic optimization problem, which is then solved with a HNN. The HNN is described by the following state transition rule (Hopfield [7]):

$$\mathbf{y}_i(k+1) = \operatorname{sgn}\left(\sum_{j=1}^{N} \hat{W}_{i,j} y_j(k) - \hat{b}_i\right), i = \operatorname{mod}_N k, \tag{8}$$

where

$$\begin{aligned} \mathbf{d} &= -\operatorname{diag}(\mathbf{W}), \\ \mathbf{W} &= -\mathbf{W} - \operatorname{diag}(\mathbf{d}), \\ \mathbf{b} &= \mathbf{b} - \frac{1}{2}\mathbf{d} \end{aligned} \tag{9}$$

Using the Lyapunov method, Hopfield [7] proved that the HNN converges to the minimum of the quadratic Lyapunov function:

$$\mathbf{y} := -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \hat{W}_{ij} y_i y_j + \sum_{i=1}^{N} y_i \hat{b}_i = -\frac{1}{2}\mathbf{y}^T \hat{\mathbf{W}} \mathbf{y} + \hat{\mathbf{b}}^T \mathbf{y}. \tag{10}$$

In order to use the HNN, the existing optimization problem is transformed into quadratic form. We give a brief summary of this approach which is fully explained in (Fogarasi et al 2012).

The original objective function (2.3) is elaborated as follows:

$$\min_{c} \sum_{i=1}^{N} w_i \left(\max\left(0, \operatorname*{argmax}_{j}\left\{C_{i,j} = 1\right\} - K_i\right)\right) \tag{11}$$

The constraints in (4) are rewritten as an optimization term later to be included into the objective function as follows:

$$\forall i : \sum_{j=1}^{L} C_{i,j} = x_i \rightarrow \min_{C} \sum_{i=1}^{N} \left(\left(\sum_{j=1}^{L} C_{i,j}\right) - x_i\right)^2 \tag{12}$$

The other set of constraints described in (5) are transformed similarly as follows:

$$\forall j : \sum_{i=1}^{N} C_{i,j} = V \rightarrow \min_{C} \sum_{i=1}^{L} \left(\left(\sum_{j=1}^{N} C_{i,j}\right) - V\right)^2 \tag{13}$$

The binary scheduling matrix **C** is mapped into a binary column vector **y** and the above three optimization terms are combined, in order to get the parameters of the HNN into the form of equation (10):

$$\mathbf{W} = \alpha \mathbf{W}_A + \beta \mathbf{W}_B + \gamma \mathbf{W}_C \in \mathcal{R}^{NL \times NL} \tag{14}$$

and

$$\mathbf{b} = \alpha \mathbf{b}_A + \beta \mathbf{b}_B + \gamma \mathbf{b}_C \in \mathcal{R}^{NL \times 1} \tag{15}$$

Note that the relative importance of the objectives is controlled by heuristic constants α, β, γ. Having this quadratic form at hand, the HNN is able to provide an approximate solution to the optimization problem in polynomial time.

In our numerical tests, the HNN recursion was repeated 1000 times with different random starting points and the best solution was used. In addition, the heuristic parameters are also adjusted during the iterations in order to provide a good balance between optimizing the objective function and also meeting the required constraints to produce a valid scheduling matrix. Furthermore, solution schedules not fully complying with the required constraints are corrected through a simple algorithm.

4 Novel Heuristic Methods

In this section, three algorithms are proposed in order to improve upon the existing methods. Unfortunately, traditional algorithms which provide fast convergence often fail to reach the global optimum. As such, the development of these novel heuristics is motivated by the desire to avoid getting stuck in local minima and still reach good quality solutions in real time by exploiting the fast convergence of the HNN. The proposed methods are based on the careful choice of the initial state or random perturbation of the initial state.

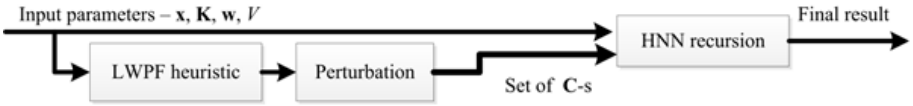


Fig. 2. Block diagram of the PSHNN method

4.1 Smart Hopfield Neural Network (SHNN)

The SHNN approach uses the result of the LFPW heuristic as input for the HNN recursion. The block diagram of the method is provided in Figure 1. In this way the HNN recursion is not being repeated several times with different starting points, therefore the method can be executed faster than the original HNN method; however repetitions due to the tuning of the heuristic parameters are still required.

The intuition behind this method is that since the value of the Lyapunov function monotonically decreases during the HNN recursion, a better starting point may converge faster and, depending on the shape of the Lyapunov surface to be minimized, could converge to a better quality minimum than a random starting point.

4.2 Perturbed Smart Hopfield Neural Network (PSHNN)

The SHNN method is faster than the original HNN solution, as it only considers a single starting point. However, as we have found through numerical simulations, it can get stuck in local minima. We can improve upon this by considering random perturbations to the "smartly" selected initial starting point. The result of the LFPW method is perturbed randomly, in order to generate a set of initial point for the HNN recursion. The block diagram of this method is shown in Figure 2.

Figure 3 provides a visual explanation as to why perturbing the initial point may have a beneficial effect when the HNN iteration gets stuck in a local minimum.

The algorithm which generates the set if inputs is described by Algorithm 1, where the function random(Θ) produces a uniformly distributed random integer value in range Θ and the function `correct(`$\mathbf{C}, \mathbf{X}$`)` is an error correction function, given by Algorithm 3. The degree of perturbation is an input parameter; however in our implementation it is proportional to the number of the columns of scheduling matrix.

The final result is the best solution provided by the HNN recursions from the set of input points generated by the algorithm.

4.3 Perturbed Largest Weighted Process First (PLWPF)

Analogously to the relationship between the SHNN and PSHNN methods, the PLWPF is the perturbed version of the LWPF method. The block diagram is shown in Figure 4.

The random perturbation is performed using Algorithm 1 and among the perturbed $\mathbf{C}$ matrices, the one which produces the lowest value for the objective function is chosen as the final result of the PLWPF method.

5 Numerical Results

In this section, the performance of the novel heuristic approaches is investigated and is compared to the performance of already known heuristics.

5.1 Simulation method

Each method outlined in Section 3 and the HNN method has been tested by extensive simulations on a large and diverse set of input parameters with the aim to characterize the algorithms empirically on scheduling problems of different sizes. The algorithms were implemented in MATLAB and tests were run in this simulation environment with randomly generated parameters such as size of jobs, cut-off times, and weights. The size of each job and its cut-off time and corresponding weight are generated as follows:

$$\mathbf{x}_i = \text{random}([1, c_1]), \tag{16}$$

$$\mathbf{K}_i = \mathbf{x}_i + \text{random}([c_1, 1.5 \cdot c_1]), \tag{17}$$

$$\mathbf{w}_i = \text{random}([1, c_2]), \tag{18}$$

where random(Θ) produces a uniformly distributed random integer value in range Θ.

In our simulations the constant c_1 was set to 10 and c_2 was set to 5. The number of processors (V) was determined as follows:

$$V = 0.25 \cdot J \tag{19}$$

The problem is expected to be solved without tardiness when the capacity is such that $V = J$. Therefore (19) ensures that there is a high likelihood of tardiness associated with the generated problem.

For each problem size (dimension of job vector), 100 different problems were generated randomly, using (16) – (19) and the results of the methods were compared in each case.

The result of LWPF was used as the initial state for SHNN. Also, this result was perturbed randomly and these perturbations were used as starting points for PSHNN. The best perturbed result is the result of PLWPF.

The HNN and PSHNN methods were repeated 1000 times for each problem with different random or perturbed starting points and the best solution was used. In addition, the heuristic parameters α, β and γ were adjusted between the simulations in order to provide a good balance between optimizing the objective function, but also meeting the required constraints to produce a valid scheduling matrix.

To adjust the heuristic parameters we used Algorithm 2.

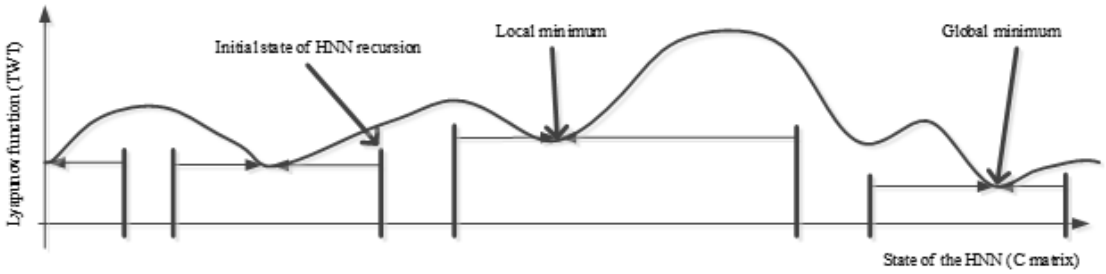


Fig. 3. Illustration of the advantage of multiple perturbed starting points in finding the global minimum

Fig. 4. Block diagram of the PLWPF method

Algorithm 1 Perturbation algorithm

perturb($\mathbf{C}_{LWPF}, N, D, \mathbf{x}$)- (Initial C, Number of outputs, Degree of perturbation, Job sizes)

$C_{RES} = \varnothing$;
for $i = 1 \rightarrow N$ **do**
  $\mathbf{C} = \mathbf{C}_{LWPF}$
  for $j = 1 \rightarrow D$ **do**
    $\{k_1, k_2, l\} \leftarrow \{random([1..J]), random([1..J]), random([1..L])\} \wedge \{\mathbf{C}_{k_1,l}, \mathbf{C}_{k_2,l}\} \leftarrow \{\mathbf{C}_{k_2,l}, \mathbf{C}_{k_1,l}\}$
  end for
  $\mathbf{C}$ = correct($\mathbf{C}, \mathbf{X}$)
  $C_{RES} = \{C_{RES}, \mathbf{C}\}$
end for
return C_{RES}

Algorithm 2 Algorithm for adjusting the heuristic parameters

Require: $\mathbf{x}, \mathbf{K}, V, e$
$\alpha \leftarrow 0.1, \beta \leftarrow 5, \gamma \leftarrow 5$
$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  $\mathbf{C}_i \leftarrow$HNN($\mathbf{x}, \mathbf{K}, V, \alpha, \beta, \gamma$)
  $\alpha \leftarrow \alpha + 0.001$
until error($\mathbf{C}_i$) $\leq e$
for $k = 1 \rightarrow i$ **do**
  $\mathbf{C}_k \leftarrow$ correct($\mathbf{C}_k$)
  $\mathbf{T}_k \leftarrow$ calculateTWT($\mathbf{C}_k$)
end for
return min($\mathbf{T}$)

Algorithm 3 Correction algorithm for the scheduling matrix produced by the HNN

correct($\mathbf{x}, \mathbf{w}, \mathbf{K}, V, \mathbf{C}$)

for $k = 1 \rightarrow L$ **do**
  while $\sum_{i=1}^{N} \mathbf{C}_{i,k} > V$ **do**
    Remove 1 from row j, where j is the row in column k with minimal weight that has $C_{j,k} = 1$
  end while
end for
for $k = 1 \rightarrow N$ **do**
  while $\sum_{i=1}^{L} \mathbf{C}_{k,i} > \mathbf{x}_k$ **do**
    Remove 1 from row k from the column l, where l is the righternmost column in row k such that $C_{k,l} = 1$
  end while
  while $\sum_{i=1}^{L} \mathbf{C}_{k,i} < \mathbf{x}_k$ **do**
    Add 1 to row k in column l, where l is the lefternmost column where 1 can be added without violating the capacity constraint V
  end while
end for
return $\mathbf{C}$

In order to achieve valid solutions in all cases by HNN, we introduced an error correction function (see Algorithm 3) in order to cut and replace the unnecessary 1-s in the scheduling matrix. In our simulations, parameter e was set to 5. All of the results presented in the following sections therefore concern valid schedules meeting the required constraints.

5.2 Average total weighted tardiness of the different methods

The first simulation compares the average total weighted tardiness provided by the algorithms for different problem sizes. For each problem size, 100 different problems were generated randomly, using (16) – (19) and the results of the methods were compared to the theoretically best schedule, obtained by exhaustive search in Figure 5 up to a problem size of 35 jobs. Simulations then were extended to problem sizes of up to 100 jobs and

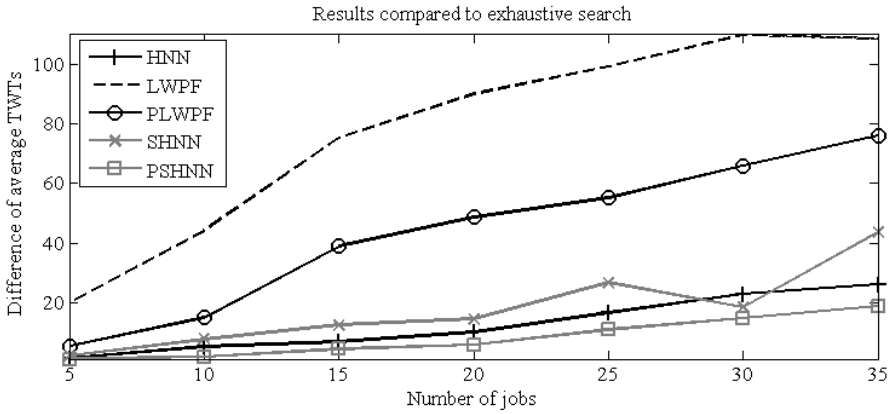


Fig. 5. The difference between the average TWT produced by each heuristic and the theoretical best schedule obtained by exhaustive search, over 100 randomly generated problems for each problem size.

the difference of the resulting average TWT of each heuristic method to the result of the PSHNN method is depicted in Figure 6. It can be observed that the best solution achieved by the PSHNN method has better average TWT for all problem sizes than any of the other heuristics. Previously, we showed that the performance of the LWPF method is, on average, worse than the HNN method for all problem sizes (Fogarasi et al 2012). However, we see that random perturbation does improve LWPF, as the solution achieved by PLWPF method is better than the solution of the original LWPF method for all problem sizes. The SHNN solution can be evaluated faster than HNN; however, its average TWT is only slightly better than the HNN method in case of small number of jobs and is significantly worse for larger problem sizes. Therefore, SHNN as a method in itself is not that impressive; however, as a stepping stone towards the PSHNN method its role is important.

We conclude that, on average, the best solution of the PSHNN outperforms the traditional solutions: the total weighted tardiness of the best PSHNN solution is 93%-97% of the HNN, 68%-96% of the LWPF; the TWT of the best PLWPF solution is 4%-95% of the original LWPF; the TWT of the SHNN solution is 77%-110% of the PLWPF. As the problem size increases, the PLWPF heuristic produce solutions which are closer to the PSHNN. The ratio between the HNN and PSHNN is relatively stable independently of the problem size.

Tab. 1. Percentage of problems in which PSHNN provides an improved solution over the next best heuristic, HNN.

Job size	
5	100%
10	100%
15	99.9%
20	99.8%
30	99.8%
40	99.9%
50	99.7%
75	99.7%
100	99.5%

In order to verify that the PSHNN consistently outperforms the other heuristics on a wide spectrum of problems, not just on a few selected problems for each given problem size, we ran a more detailed numerical experiment. In this case, we investigated 500 randomly generated problems for each problem size and computed the percentage of times the PSHNN produced a better solution than HNN, the next best method. Table 1 shows the result which quite convincingly proves the general applicability of the PSHNN to different problem types. Across the spectrum of problem domain, the ratio is higher than 99.2% for all investigated job sizes.

5.3 Parallel implementation of the algorithm

In this section, for further speed-up we investigate and summarize a possible implementation of the HNN on GPGPU, specifically on Fermi architecture (Kirk, D. B. et al. 2010). Table 2 contains the theoretical order of convergence of the algorithms as a function of the length of the input parameters.

Tab. 2. Theoretical runtime of the algorithms

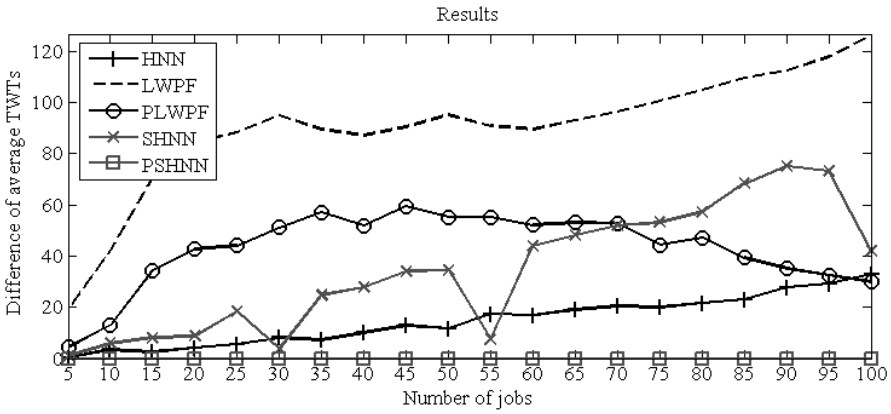
Algorithm	Theoretical order of convergence
LWPF	$O(L \cdot N)$
HNN	$O(L^2 \cdot N^2)$ [6, 7]

Although the number of repetitive iterations does not affect the theoretical order of convergence, the actual runtime of the algorithms may increase significantly. In order to reduce overall runtimes, each HNN run within the PSHNN method should be parallelized, resulting in multiple HNN runs from perturbed starting points taking the same time as a single HNN run. Furthermore, Liang (2011) introduced a highly parallelized implementation of the HNN, which allows for significant speedup of each HNN run. More formally, the total runtime T of the algorithm is given by:

$$T = R_{init}(R_{heur}t) \tag{20}$$

where R_{init} denotes the number of repeats of the algorithms from different initial starting points, R_{heur} denotes the number of repeats of the algorithm with different heuristic parameter (alpha, beta, gamma) settings and t denotes the runtime of a single non-parallelized HNN run. For example, assuming a 200×200 sized

Fig. 6. The difference between the average TWT produced by each heuristic and the average TWT produced by the best method, PSHNN over 100 randomly generated problems for each problem size.



W weight matrix corresponding to 30 jobs to be scheduled, with $R_{init} = 500$, $R_{heur} = 100$, the estimated time of execution on a 3GHz CPU processor is 50 seconds running optimized MATLAB code.

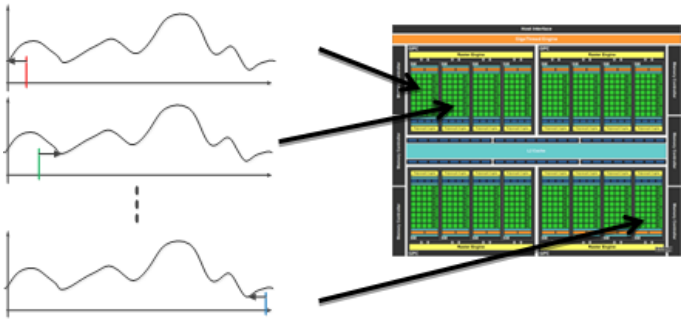


Fig. 7. Illustration of how each run of the perturbed starting point may be parallelized in a GPGPU infrastructure.

Since each iteration is independent of other iterations, it can be executed independently and in parallel. Using an existing parallel HNN implementation, we extend the parallelism on nVidia Fermi architecture as follows (Farber [4]):

- The **W** matrix is sparse (5-10 % of values are nonzero, and there are patterns amongst the nonzero values), therefore in case of large **W** matrix one HNN iteration can be executed by a stream multiprocessor (SM) using its local (shared) memory.
- The independent HNN iterations can be executed on separate SM-s.

As such, the execution time of the PSHNN algorithm can be significantly decreased on Fermi multicore architecture, thus we can achieve a better solution than the other heuristics within the same execution time. The idea of the parallelization is illustrated in Figure 7 and the execution times are shown in Figure 8.

5.4 Practical case study on large problem size

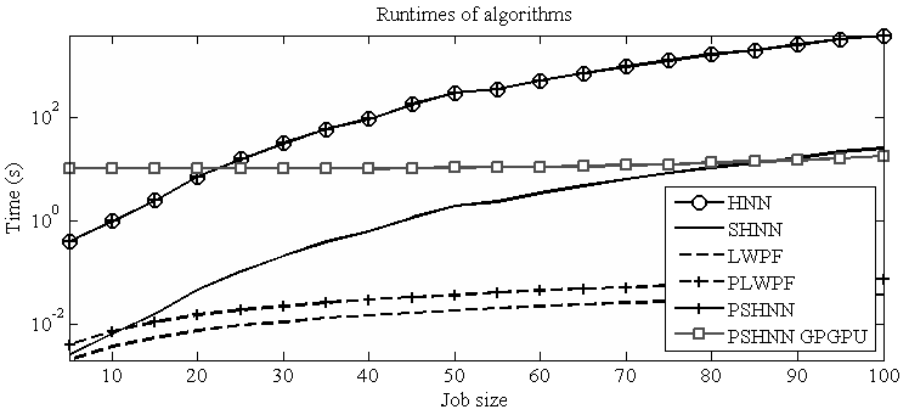
A common task arising in computational finance is the scheduling of a set of valuation and risk sensitivity calculations which need to be run overnight based on the close-of-market market observables. Complex derivative transactions cannot, in general, be priced analytically and require Monte Carlo simulations for their valuation. In the specific example studied here, we consider the risk calculations associated with nearly 100 different portfolios, yielding altogether 556 jobs of an average size of 792 seconds to be scheduled. Each job has associated with it a cutoff time, by which time risk calculations need to finish in order for the results to be incorporated in the nightly firmwide Value at Risk (VaR) calculation and in order to have the sensitivity figures available for the trading desk at market open. Furthermore, each job has a priority associated with it which is assigned based on the relative amount of risk carried in that portfolio. We obtained specific runtime, deadline and priority data for an actual overnigh scheduling problem for traded portfolios at Morgan Stanley, one of the largest financial institutions in the world.

Having run each of the algorithms on this set of real data, Table 3 summarizes the results. As seen, PSHNN outperforms the next best method, PLWPF by 4.7% in terms of TWT, HNN by 7%, LWPF by 10% and EDD by a striking 48%. Another way to interpret the results is that PSHNN essentially completes all jobs with top priorities (8,9,10) on time, whilst PLWPF has a total tardiness of around 53 minutes (4 average sized jobs) on these top priority jobs although it completes the less important jobs more quickly.

6 Conclusions and Directions of Future Research

In this paper, we studied the NP-hard problem of scheduling jobs with given relative priorities and deadlines on identical machines, minimizing the TWT measure. In our previous work, we showed that the HNN approach is a heuristic which consistently outperforms other benchmark heuristics such as the EDD and WSPT methods. In this paper, we improved our previous work by demonstrating that random perturbations to the intelligently selected starting point significantly improves the quality of the solution of the HNN approach. Furthermore, we showed that applying random perturbations to the deterministic LWPF heuristic also provides an improvement. We studied the implementation details of the random perturbations and the HNN method to show that the methods remain applicable in real-time settings

Fig. 8. Runtimes of the different algorithms, as a function of job size.



Tab. 3. Table of total (weighted) tardiness for jobs of each weight, provided by the different methods for the Morgan Stanley scheduling problem

Total Weighted Tardiness

Weights	3	4	5	6	7	8	9	10	SUM	Increment to PSHNN
PSHNN	4401	11116	4020	1620	1092	8	0	0	22257	0 %
PLWPF	3513	9624	5130	1788	490	312	2304	190	23351	5 %
HNN	4404	11040	4735	1824	1092	456	468	0	24019	7 %
LWPF	4404	11140	5470	2472	1183	40	0	0	24709	10 %
EDD	4401	9940	1770	636	1134	464	22752	1430	42527	48 %

Total Tardiness (Minutes)

Priority	3	4	5	6	7	8	9	10	SUM
PSHNN	244.5	463.2	134.0	45.0	26.0	0.2	0.0	0.0	912.8
PLWPF	195.2	401.0	171.0	49.7	11.7	6.5	42.7	3.2	880.8
HNN	244.7	460.0	157.8	50.7	26.0	9.5	8.7	0.0	957.3
LWPF	244.7	464.2	182.3	68.7	28.2	0.8	0.0	0.0	988.8
EDD	244.5	414.2	59.0	17.7	27.0	9.7	421.3	23.8	1217.2

for scheduling large number of problems, PSHNN yielding a 5% improvement over the next best method, PLWPF and 7% improvement over regular HNN.

As for directions for further research, more formal methods for the selection of alpha, beta and gamma parameters could be investigated in order to further improve performance and bounds on the performance of these methods could be established, in relation to the theoretical optimum.

Aknowledgement

The work reported in the paper has been developed in the framework of the project "Talent care and cultivation in the scientific workshops of BME" project. This project is supported by the grant TAMOP – 4.2.2.B-10/1–2010-0009.

References

1 **Akyol DE, Bayhan GM**, *Multi-machine earliness and tardiness scheduling problem: an interconnected neural network approach*, Int. J. of Adv. Manuf. Technol., **37**, (2008), 576–588.

2 **Brucker P**, *Scheduling Algorithms*, Springer; New York,USA, 2007.

3 **Du J, Leung JY-T**, *Minimizing total tardiness on one machine is NP-hard*, Math. Oper. Res, **15**(3), (1990), 483–495.

4 **Farber R**, *CUDA Application design and development*, 1st edn., Morgan Kaufmann, 2011.

5 **Fogarasi N, Tornai K, Levendovszky J**, *A Novel Hopfield Neural Network Approach for Minimizing Total Weighted Tardiness of Jobs Scheduled on Identical Machines*, Acta Univ. Sapientiae, Informatica, **4**(1), (2012), 48–66.

6 **Haykin S**, *Neural Networks and Learning Machines*, Prentice Hall, 2008. 3rd Edition.

7 **Hopfield JJ**, *Neural networks and physical systems with emergent collective computational abilities*, Proceedings of the National Academy of Sciences of the USA, (1982).

8 **Kirk DB, Hwu WW**, *Programming massively parallel processors: A hands-on approach*, Morgan Kaufman., 2010.

9 **Liang L**, *Parallel implementations of Hopfield neural networks on GPU*, 2011.

10 **Levendovszky J, Olah A, Tornai K, Treplan G**, *Novel load balancing algorithms ensuring uniform packet loss probabilities for WSN*, 2011 IEEE 73rd Vehicular Technology Conference, (2011).

11 **Maheswaran R, Ponnambalam SG, Nithin SD, Ramkumar AS**, *Hopfield Neural Network Approach for Single Machine Scheduling Problem*, Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems, (2004).

12 **Naderi B, Zandieh M, Shirazi MAHA**, *Modeling and scheduling a case of flexible flowshops: Total weighted tardiness minimization*, Computers & Industrial Engineering, **57**(4), (2011), 1258–1267.

13 **Pinedo M**, *Scheduling – Theory, Algorithms and Systems*, Springer; New York, 2008.