A Neurodynamic Approach for Real-Time Scheduling via Maximizing Piecewise Linear Utility

Zhishan Guo, *Student Member, IEEE*, and Sanjoy K. Baruah, *Fellow, IEEE*

***Abstract*— In this paper, we study a set of real-time scheduling problems whose objectives can be expressed as piecewise linear utility functions. This model has very wide applications in scheduling-related problems, such as mixed criticality, response time minimization, and tardiness analysis. Approximation schemes and matrix vectorization techniques are applied to transform scheduling problems into linear constraint optimization with a piecewise linear and concave objective; thus, a neural network-based optimization method can be adopted to solve such scheduling problems efficiently. This neural network model has a parallel structure, and can also be implemented on circuits, on which the converging time can be significantly limited to meet real-time requirements. Examples are provided to illustrate how to solve the optimization problem and to form a schedule. An approximation ratio bound of 0.5 is further provided. Experimental studies on a large number of randomly generated sets suggest that our algorithm is optimal when the set is nonoverloaded, and outperforms existing typical scheduling strategies when there is overload. Moreover, the number of steps for finding an approximate solution remains at the same level when the size of the problem (number of jobs within a set) increases.**

***Index Terms*— Neurodynamic optimization, NP-hard problem, overloaded job set, real-time scheduling, recurrent neural network (RNN), utility maximization.**

I. Introduction

REAL-TIME systems are widely applied in vehicles, mobile phones, satellites, traffic management, and so on, whose correctness depends on the temporal aspects as well as the functional aspects. Examples of real-time systems include digital command and control, signal processing, and telecommunication systems [1]. A key characteristic of a real-time system is the level of its consistency, concerning the amount of time it takes to accept and complete an application's task.

A. Real-Time Scheduling

Timing requirements in real-time systems are often modeled as deadlines. If a schedulable activity (e.g., a piece of code-job) executes and completes before its assigned deadline, the deadline is met (and otherwise is missed). In hard real-time systems [2], any deadline miss is considered equivalent to a catastrophic failure. In contrast, failure to meet a deadline in soft real-time systems only impacts the quality of service (QoS).

Although highly predictable, hard real-time systems are built under pessimistic assumptions to cope with worst case scenarios, and thus are not sufficiently flexible to adapt to dynamic situations. Soft real-time systems are built to reduce resource consumption, to tolerate overloads, and to adapt to system changes. Those are more suited to the novel applications of real-time technology, such as multimedia systems, monitoring apparatuses, telecommunication networks, mobile robotics, virtual reality, and interactive computer games [2].

Different kinds of targets have been set in soft real-time scheduling, such as minimizing the number or frequency of missed deadlines, minimizing the tardiness bound, and the maintenance of fairness or stability. Soft real-time scheduling is commonly used to support quality of service (QoS) by combining predictable resource allocation (e.g., proportional sharing and reservation) and real-time scheduling algorithms [3] (e.g., earliest deadline first (EDF) [4]).

B. Utility Maximization

In addition to restrictions on the system induced by timing requirements, other constraints may also influence the deployment and design of real-time systems. For example, the value associated with a job reflects its importance with respect to the other jobs. However, the actual value of a job also depends on the time at which the job is completed in real-time systems. Utility functions offer a general way to describe the complex constraints of real-time and cyber-physical systems [5]. Several utility accrual algorithms have been proposed to solve energy-efficient scheduling problems [6], [7].

Take mixed-criticality scheduling (see [8] for a review) as another example, where the functionalities of different degrees of importance are implemented upon a common platform. Such a mixed-criticality property is becoming more and more common in embedded systems, as is evidenced by industry-wide initiatives, such as integrated modular avionics for aerospace and automotive open system architecture for the automotive industry. Criticality levels may also be differentiated under certain utility functions (one possible choice of such utility functions is described in Section II with more details).

Manuscript received August 15, 2014; revised May 18, 2015 and July 28, 2015; accepted July 30, 2015. Date of publication August 28, 2015; date of current version January 18, 2016. This work was supported in part by the National Science Foundation under Grant CNS 1016954, Grant CNS 1115284, Grant CNS 1218693, Grant CPS 1239135, Grant 1409175, and Grant 1446631, in part by the Air Force Office of Scientific Research under Grant FA9550-14-1-0161, in part by the Army Research Office under Grant W911NF-14-1-0499, and in part by General Motors Corporation.

The authors are with the Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599 USA (e-mail: zsguo@cs.unc.edu; baruah@cs.unc.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2015.2466612

C. Neurodynamic Optimization

Since Tank and Hopfield's pioneering work [9], which constructed a neural network to solve linear programming problems, recurrent neural networks (RNNs) for optimization and their engineering applications have been widely investigated in the past decades. For example, the Lagrangian network for solving nonlinear programming problems with equality constraints [10], the deterministic annealing network for convex programming [11], the primal–dual neural network for convex quadratic programming [12], and a generalized neural network for nonsmooth nonlinear programming problems [13] were developed.

More recently, several RNNs with discontinuous activation functions were proposed for solving nonlinear optimization problems with various kinds of constraints, see [14]–[20] (this list is by no means exhaustive). In particular, Liu and Wang [21] proposed a RNN with hard-limiting activation function for constrained optimization with piecewise linear objective functions, and proved its finite-time convergence.

D. Related Work

Scheduling problems are often highly intractable (NP-hard), and traditional algorithms are approximate yet often with high time complexity [22]. The Hopfield neural network is commonly applied to obtain an optimal or suboptimal solution in various different scheduling applications, such as the traveling salesman problem (which is a typical NP-hard discrete combinatorial optimization problem) [9]. RNNs can be easily implemented by hardware circuits or by parallel algorithms due to its structural feature, and thus may significantly shorten the online scheduling computation cost in real-time systems. Meanwhile, offline scheduling strategy (typically with high time complexity) could be solved by RNNs in microseconds (or even shorter), and thus becomes online applicable. This is to say, with the help of RNNs, schedulers may now handle unexpected changes well enough (or even optimally) during run time efficiently. Another key feature for many RNN systems is the convergence time barely relies on the size or dimension of the problem. Thus, RNNs may be very promising approaches in solving the scheduling problems of larger scale platform, e.g., CPU-sharing grid and cloud computing center.

There have been some attempts in solving real-time scheduling problems through neural networks. Cardeira and Mammeri [23], [24] made some initial attempts in using Hopfield neural networks to solve real-time scheduling problems. Their algorithm is approximate, but has a remarkable convergence speed due to the highly parallel nature of the searching process. Silva *et al.* [25] presented a systematic procedure to map the scheduling problem onto a neural network, which overcomes the possibility of RNN solutions, leading to unfeasible scheduling trouble due to suboptimality. Based on inhibitor neurons, Huang and Chen [26], [27] applied the normalized mean field annealing technique to solve a known NP-hard multiprocessor scheduling problem, and further dealt with the cases with time constraints (execution time and deadline), no process migration, and limited resources. More recently, Baruah *et al.* [28] extended the existing approaches to consider platform heterogeneity, where the number of neurons is reduced by a factor of more than two, and the number of task migrations is reduced and can be limited compared with the PFair algorithm (known as an optimal solution in the context of homogeneous architecture).

Unfortunately, existing work on using RNNs for real-time scheduling deals with very specific problems under certain assumptions. Most of them are based on aged neural network models, and are hard to extend for more general scheduling purposes. Thanks to the continuous efforts in the computational intelligence community for the past two decades, RNN-based optimization approaches are much more powerful and efficient nowadays. Such tremendous improvements in the capability of RNN models somewhat trigger us to revisit the intersection area of neurodynamic optimization and real-time scheduling. This paper initiates a more general way (with the piecewise linear utility function) of expressing scheduling-related problems, and provides further promising directions combining neurodynamic optimization with real-time scheduling.

E. Contribution and Paper Organization

In this paper, we target the real-time utility maximization scheduling problem on a uniprocessor platform. We show its NP-hardness, and provide a neurodynamic optimization approach to approximately solve it. Unlike any of the previous works, the RNN model we apply has a much simpler structure (with only one layer and $O(n^2)$ neurons, where n is the number of jobs), and is finite-time global converging. Several typical choices of utility functions are discussed, each of which corresponds to a focusing criterion in real-time scheduling.

The remainder of this paper is organized as follows. In Section II, we formally describe the system and workload models assumed in this paper, and prove the NP-hardness of the optimization problem, with two demonstration examples. In Section III, a neurodynamic optimization approach is described to approximately solve the problem. The experimental study of the proposed approach and a comparison over classic scheduling strategies are provided in Section IV. Section V further analyzes the approximation ratio of the proposed method, and discusses possible model extensions. Finally, Section VI concludes this paper and points out some further research directions.

II. Model and Problem Description

In this paper, we consider a workload model consisting of independent jobs. Finite collections of independent jobs may arise in the frame-based approaches of real-time scheduling: the frame represents a collection of periodic jobs sharing a common period (or the frame is extended to the least common multiple of different tasks' periods and every job of each task is represented individually within the frame), and the entire frame is repeatedly executed.

In our model, a real-time workload consists of the basic units of work known as jobs, where each job J_i is characterized

by a 3-tuple of parameters or functions: a release time a_i, a worst case execution time (WCET) estimation c_i [29], and a utility function $\mu_i(t) : [0, +\infty) \rightarrow \mathbb{R}$ (also known as the cumulative value in some early studies).[1] Given any time instant t that a certain job J_i finishes its execution, the utility function $\mu_i(t)$ returns the benefit (reward) a given schedule achieves. Note that by making certain choices of utility functions, many widely used scheduling concerns may be represented, such as the deadline d_i (details are discussed in Section II-A).

A schedule ζ specifies the allocation of all jobs to the whole time interval, indicating which job should be executed at any given instant. A job J_i can only be executed on or after it is released (when $t \geq a_i$), and may require as much as a cumulative execution of length c_i to be finished. Here, we assume that the job preemption is permitted with zero cost, i.e., any job can be temporarily hanged up during execution and retrieved afterward immediately. Given a schedule ζ, the instant $y_{\zeta,i}$ that each job finishes its execution can be derived, and the goal is to maximize the overall utility accordingly

$$\max_{\zeta} \sum_{i=1}^{n} \mu_i(y_{\zeta,i}). \tag{1}$$

This is a very generalized form of objective, and different detailed deductions will be provided later. Here, we first introduce several important concepts that are often related to scheduling concerns and objectives.

Definition 1 (Response Time): Given a schedule ζ, the response time of a job J_i (with release time a_i and deadline d_i) is given by $y_i - a_i$, where y_i is the completion time under such schedule.

Definition 2 (Tardiness): Given a schedule ζ, the tardiness of a job J_i (with release time a_i and deadline d_i) is given by $\max\{0, y_i - d_i\}$, where y_i is the completion time under such schedule.

If a deadline d_i is assigned, then we may calculate the load of a given set.

Definition 3 (Load): The load of a given job set $\{J_i\} = \{a_i, c_i, d_i\}$ can be represented as

$$L(\{J_i\}) = \max_{t_s, t_f | t_s < t_f} \frac{\sum_{i | a_i \geq t_s, d_i < t_f} c_i}{t_f - t_s} \tag{2}$$

where t_s and t_f are the two time instants along the scheduling time line.

Intuitively, load is given by the maximum execution requirement density over any continuous scheduling interval $[t_s, t_f)$. Any job set with load less than 1 can be optimally scheduled (e.g., by EDF) when deadline meeting is the only goal. For other objectives, such as response time minimization, more complicated utility functions need to be developed.

The average response time often serves as one of the key benchmarks in real-time systems. While in soft real-time scheduling or overloaded (set with load greater than 1) case, tardiness often serves as a measurement of QoS or scheduling performance, and the searching for bounded (and even minimized) tardiness scheduling strategies under various platforms and conditions has been another hot topic in real-time scheduling community for the past decade [3], [30], [31].

[1] Actually, jobs can never be finished before they are released, and thus it is fine to give $\mu_i(t)$ any definition within the range $[0, a_i)$ upon constraints that will be provided later on.

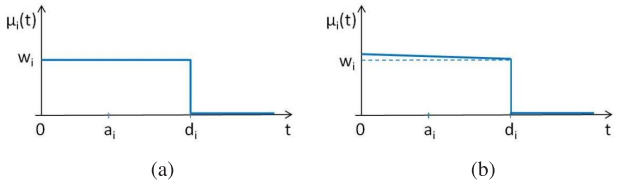


Fig. 1. Utility function choices for a given job $J_i = \{a_i, c_i, d_i\}$, where w_i are weighting parameters of the utility $\mu_i(\cdot)$. (a) Step function. (b) Weighted step function.

A. About the Utility Function

Unlike previous work on utility-based scheduling, here we introduce separate utility functions for each job (instead of a generalized one for the whole set). Compared with the traditional job model in real-time scheduling, the additional concept utility has very broad applications. Although we restrict the utility function to be piecewise linear in this paper, a large amount of typical scheduling requirements with respect to deadline meeting, response time minimization, mixed-criticality scheduling, energy saving, and QoS maximization can be translated into this model. Another reason to replace the deadline with a utility function is due to the ongoing discussion on hard versus soft real-time scheduling-the release time and the WCETs may serve as constraints that cannot be violated, while deadlines may not. Many systems may be tolerant to missing some deadlines, and certain benchmarks may be achieved by maximizing particular utility functions.

In this section, we discuss some reasonable choices (as shown in Fig. 1) of utility functions, and describe their relationships to different real-time scheduling concerns.

First of all, we may set the utility function as a step function with d_i as the threshold, i.e., $\forall i, \mu_i(t) = u(d_i - t)$, where the step function $u(\cdot)$ is defined as $u(x) = 1$ if $x \geq 0$, and $u(x) = 0$ otherwise. Thus, the scheduling problem is reduced into traditional-independent job set scheduling, where scheduling strategies, such as EDF, have been proved to be optimal [4]. Note that, under the assumption that $w_i = 1$ for all i, finishing each job on time will gain the same amount of profit onto the total utility, which is maximized if and only if all deadlines are met for a given schedule. In the case where jobs are assigned different values, w_i's are no longer the same, and the problem becomes more complicated (Theorem 1 in Section II-B).

Moreover, if we slightly increase the (absolute values of) slopes of utility functions before the deadlines, as shown in Fig. 1(b), the response time of each job is also considered. As far as the additional reward of finishing jobs early does not dominate the whole utility, we are minimizing not only the tardiness, but also the total response time, which is a more general problem than the previous one. Similar to previous cases, jobs that are more important (e.g., more critical to safety

consumption) can be assigned a larger weight in order to be recognized during the optimization process.

B. Problem Description

Given a finite collection of jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ with specified release time, WCETs, and utility functions, we seek to determine an optimal schedule ζ^* that maximizes the total utility. In this section, we introduce the necessary math notations and describe this problem formally.

Similar to [32] and [33], let $t_1, t_2, \ldots, t_{k+1}$ denote the at most $2n$ distinct values for the release date and deadline parameters of the n jobs, in increasing order ($t_j < t_{j+1}$ for all j). These release dates and deadlines partition the whole scheduling window $[\min_i\{a_i\}, \max_i\{d_i\})$ into k intervals, which will be denoted by $I_1, I_2, \ldots, I_k$, with I_j representing the interval $[t_j, t_{j+1})$.

A scheduling table offers a very straightforward way of representing a schedule ζ. We define $n \times k$ nonnegative variables $x_{i,j}$, $1 \le i \le n; 1 \le j \le k$, with $x_{i,j}$ denoting the amount of execution assigned to job J_i in the interval I_j, in the scheduling table that we are seeking to build.

In addition to the nonnegative constraints for all variables

$$x_{i,j} \ge 0 \quad \forall i, j, \ 1 \le i \le n, \ 1 \le j \le k \tag{3}$$

the following k constraints specify the capacity constraints of the intervals:

$$\left(\sum_{i=1}^{n} x_{i,j}\right) \le t_{j+1} - t_j \quad \forall j, \ 1 \le j \le k. \tag{4}$$

1) Hard Deadlines and Overload Conditions: An ideal scheduler would not only complete all jobs in a nonoverloaded environment, but also minimize the overall damage to the system performance in the presence of overload. When overload occurs, a scheduling algorithm must discard some tasks in a way that maximizes the overall value of the system.

In hard real-time scheduling, the profit of executing any job beyond its deadline always remains zero. Thus, once a deadline is missed, no further execution to that job is necessary. Mathematically speaking, the expression $\sum_{j|a_i \le t_j < d_i} x_{i,j}$ gives us the cumulative execution of a given job J_i within its scheduling window $[a_i, d_i)$. This is the amount that needs to be compared with the WCET to see whether such scheduling table X guarantees to finish the job J_i on time.

With the help of the utility function, the benefit can be expressed by $\mu_i(y_{\zeta,i}) = u(\sum_{j|a_i \le t_j < d_i} x_{i,j} - c_i)$, where $u(\cdot)$ is the (increasing) step function with 0 as the threshold. As a result, given the scheduling table X, the objective (1) can be specifically expressed by

$$\max_X \sum_{i=1}^{n} u\left(\sum_{j|a_i \le t_j < d_i} x_{i,j} - c_i\right). \tag{5}$$

2) Mixed Criticality: In mixed-criticality systems, jobs are given different importance levels. Each job is assigned a criticality level χ_i in a widely recognized mixed-criticality model. In the two-level subcase, some jobs J_i (with $\chi_i = \text{HI}$) are defined to be more important than the others (with $\chi_i = \text{LO}$).

Under our utility model, the profit for finishing each job should no longer be the same. Mixed-criticality scheduling problems can be equivalently transformed into our model by multiplying coefficients $w_i \in (0, 1)$ to the objective to differentiate the values (importance levels) of the jobs

$$\max_X \sum_{i=1}^{n} w_i \cdot u\left(\sum_{j|a_i \le t_j < d_i} x_{i,j} - c_i\right) \tag{6}$$

while adding the following constraint:

$$w_i > \sum_{j|\chi_j = \text{LO}} w_j \quad \forall i, \ \chi_i = \text{HI}. \tag{7}$$

Intuitively, inequality (7) guarantees that the profit for meeting any (single) HI-criticality deadline is greater than meeting all LO-criticality deadlines, which is the sole of mixed-criticality scheduling. Note that the key to the mixed-criticality scheduling problem is meeting more important deadlines while discarding some of the rest when necessary.

3) NP-Hardness: We now show that, even when constraint (7) is ignored, this problem (under the overloaded case) is NP-hard.

Theorem 1: Optimizing (6) under constraints (3) and (4) is NP-hard.

Proof: This hardness is demonstrated by reducing the (0-1) knapsack problem [34], which is known to be NP-complete, to our problem. The knapsack problem is defined as follows. Given n items with the values $\{v_i\}$ and the weights $\{\omega_i\}$ (both positive), and the maximum weight $U > 0$ that we can carry, determine whether each item is included or not (denoted by the variable $y_i \in \{0, 1\}$), so that the total weight is no greater than the given limit ($\sum_{i=1}^{n} \omega_i y_i \le U$), and the total value ($\sum_{i=1}^{n} v_i y_i$) is maximized.

From a given instance of the knapsack problem $\{\{v_i\}, \{\omega_i\}, U\}$, we build an instance of our problem $\{\{a_i\}, \{c_i\}, \{d_i\}, \{w_i\}\}$ (in polynomial time) as follows.

1) For each i, $a_i \leftarrow 0$ and $d_i \leftarrow U$.
2) For each i, $c_i \leftarrow \omega_i$ and $w_i \leftarrow v_i$.

We now show that, by setting $y_i \leftarrow u(x_i - c_i)$, an optimal schedule x to our problem can be transformed into a solution of the knapsack problem. Since all jobs share the same release time and deadline, there is only one interval $[0, U)$ in our problem. Moreover, inequality (4) is equivalent to $\sum_{i=1}^{n} \omega_i y_i \le U$, and both the problems share the same objective. Under such transformation, $y_i \in \{0, 1\}$ denotes whether item J_i should be carried.

For the other way around, given an optimal solution y of the knapsack problem, setting $x_i \leftarrow c_i$ when $y_i = 1$, and 0 otherwise, leads to a feasible and optimal schedule.

It is now evident that $\{y_i\}$ is an optimal solution to the knapsack problem if and only if the corresponding $\mathbf{x}$ is an optimal solution to the constructed uni-interval scheduling problem. ■

III. Neurodynamic Approach

The real-time scheduling problem is NP-hard, and existing algorithms either provide weak approximate solutions or are too complicated to meet real-time requirements. To overcome such a difficulty, in this section, we introduce a neurodynamic approach to solve the above mentioned real-time scheduling problem. Since the scheduling problem is NP-hard, we first need to modify the objective function into a concave one, so that the dynamic system converges to the global optima. Given a vectorizing stationary point of the RNN system, transformation needs to be done to get a scheduling table (matrix), and the accurate executions (to all jobs on each interval) can be derived accordingly.

A. RNN Model

A typical neurodynamic approach uses gradient information as a guide to the stationary point(s), while projecting the search process onto the feasible region to deal with linear constraints. In [21], an RNN model was proposed for the following linear inequality-constrained minimization with piecewise linear objective functions:

$$\min_{\mathbf{x}} \sum_i \varphi_i \left(\sum_j C_{i,j} x_i - f_i \right) \tag{8}$$

$$\text{s.t.} \sum_j a_{k,j} x_j \le b_k \quad \forall k \tag{9}$$

where $C_{i,j}, f_i, a_{k,j}, b_k \in \mathbb{R}$, and $\varphi_i(x) : \mathbb{R}^{\|x\|} \to \mathbb{R}$ is assumed to be piecewise linear and defined as: for $l_i, h_i \in \mathbb{R}$ such that $\varphi_i(x) = l_i x$ when $x \ge 0$ and $\varphi_i(x) = h_i x$ otherwise.

Given the description of our targeting scheduling problems in Section II, it can be approximately transformed into this RNN-friendly model, where

$$A = \begin{bmatrix} I_k & I_k & \cdots & I_k \\ I_k & O_k & \cdots & O_k \\ O_k & I_k & \cdots & O_k \\ \vdots & \vdots & \ddots & \vdots \\ O_k & O_k & \cdots & I_k \end{bmatrix} = [a_{k,j}]_{(k+nk)\times(nk)} \tag{10}$$

$$\begin{aligned} \mathbf{b} &= [t_2 - t_1, t_3 - t_2, \ldots, t_{k+1} - t_k, \mathbf{0}_{nk}]^T \\ &= [b_k]_{(k+nk)\times 1} \end{aligned} \tag{11}$$

$$C_{i,j} = \begin{cases} 1, & \text{if } a_i \le t_j < d_i \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

$$\forall i, \ \ f_i = c_i \tag{13}$$

where I_k is the k by k identical matrix, O_k is the k by k zero matrix, and $\mathbf{0}_{nk}$ is a 1 by nk zero vector. Discussions on determining the two gradient information sets $\{l_i\}$ and $\{h_i\}$ are discussed in Section III-B before the proposed method is described in detailed steps.

The relationship of the variable $\mathbf{x}$ in the optimization problem and the scheduling table (matrix) X[2] is

$$\forall i, j, \ \ X_{i,j} = x_{(i-1)k+j}. \tag{14}$$

[2]The scheduling table X assigns each job J_i an execution length of $X_{i,j}$ to each interval I_j, based on which execution can be directly performed.

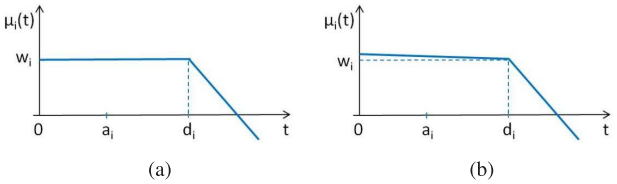


Fig. 2. Approximate utility functions for a given job $J_i = \{a_i, c_i, d_i\}$, where w_i are weighting parameters of the utility $\mu_i(\cdot)$. (a) Piecewise linear function. (b) Weighted piecewise linear function.

Under such transformation, the following RNN model [21] can be applied to solve the scheduling problem:

$$\epsilon \frac{d\mathbf{x}}{dt} = C^T g_{[l,h]}(C\mathbf{x} - \mathbf{f}) - \sigma A^T g_{[0,1]}(A\mathbf{x} - \mathbf{b}) \tag{15}$$

where $C = \{C_{i,j}\}, \mathbf{f} = \{f_i\}$, ϵ and σ are the positive scaling parameters, and $g_{[u,v]}(x)$ is given by: $g_{[u,v]}(x) = u$ when $x \ge 0$, and $g_{[u,v]}(x) = v$ otherwise.

With the hard-limiting activation function $g_{[u,v]}(\cdot)$, such RNN system is proved [21] to be stable in the sense of Lyapunov and globally convergent to an optimal solution in finite time with σ satisfying

$$\sigma \ge \frac{\max_{\xi | \xi \in \Pi_{i=1}^{p} [l_i, h_i]} \|C^T \xi\|_2}{\min_{\gamma | \forall i, \gamma_i \in [0,1]; \exists j, \gamma_j = 1} \|A^T \gamma\|_2}. \tag{16}$$

The corresponding circuit architecture of the neural network can be found in [21, Fig. 1], where the neural network has a one-layer structure with $\|x\| = n \times k$ neurons.

B. Scheduling Algorithm

Due to the nonconvexity of the step function described in Fig. 1(a), it is often very difficult or time-consuming to find the global optimum. In overloaded conditions where there exists no schedule that meets all deadlines, the problem of choosing the right jobs at the right time for optimal total utility has been proved to be NP-hard (Theorem 1 in Section II-B).

The optimization problem described in Section II is not only NP-hard, but also lack of informational gradient over the search space (time line). In this paper, we substitute the step functions [Fig. 1(a) and (b)] into piecewise linear, Lipschitz continuous, and most importantly concave ones [e.g., Fig. 2(a) and (b)]. By choosing such piecewise linear concave objective function, we are transforming the original NP-hard problem into a constrained convex optimization problem that can be solved with the gradient-based dynamic systems like (15).

Under such kinds of utility functions, the benefit of finishing a job drops linearly after its deadline, and may become negative. In the case where all jobs share the same slope after deadlines, the cumulative tardiness is minimized when maximizing the utility.

Since we are maximizing the utility in the scheduling problem while the RNN model is minimizing its piecewise linear objective, a negative sign need to be applied for slope transformation, i.e., $l_i = -w_i$. Selection of h_i is based on the scheduler's concern: in the case when response time is considered, a negative h_i with the same absolute value of

TABLE I
PARAMETERS OF JOBS CONSIDERED IN EXAMPLE 1

Job	a_i	d_i	c_i	w_i
1	0	3	2	0.699
2	0	3	1	0.198
3	2	9	5	0.013

the slope in Fig. 2(a) can be assigned; while in other cases, h_i may be simply set as 0.

Note that such change to the objective function does not affect the maximum possible value, and thus the two problems with different objectives are equivalent when the system is not overloaded-there exists a schedule to have all deadlines met. For the overloaded case, the new objective naturally penalizes the scheduler according to the unfinished amount of each job, and thus may lead to an appropriate approximate solution. Some further discussions on this are provided in Sections IV and V.

Given a job set $\{J_i\}$ and the weights $\{w_i\}$, our scheduling algorithm performs in the following steps.

1) Calculate the number of intervals k according to the release time $\{a_i\}$ and the deadlines $\{d_i\}$.
2) Calculate A, b, C, f, l, and h according to the described transformation rules.
3) Apply the RNN model (15), with a large enough σ satisfying (16) and an initial point of $\mathbf{x}_0 \leftarrow \mathbf{0}$.
4) Record the state variable x^* after the system converges.
5) Transform $\mathbf{x}^*$ back into a scheduling table X^* (which is an $n \times k$ matrix) according to (14), and perform the executions accordingly.

During execution, any order can be assigned within each interval, and will result in the same utility-we specifically assume it to be EDF in the experiments.

The following randomly generated example shows how our RNN-based approach works out intuitively.

Example 1: We will consider a nonoverloaded job set consisting of three jobs with parameters as depicted in Table I, where different jobs are randomly assigned a penalty coefficient w_i-with larger value denoting its higher importance. EDF may serve as an optimal schedule, since the job set is not overloaded.

Fig. 3 shows the convergence process of the RNN model. Since the RNN is proved to be globally convergent, initial states do not matter, and are set to be 0 for convenience in all experiments. Here, in order to better measure how fast the system may converge for a digital circuit system, we apply a discrete version of the model, so that the time line is shown as number of steps. Normally, each step would take <1 μs to finish, and thus the whole optimization process takes a period within the millisecond level.

The solution $\mathbf{x}^* = [1.25, 0.75, 0, 0.75, 0.25, 0, 0, 0, 5]^T$ (as a vector) needs to be transformed into a scheduling table matrix

$$X^* = \begin{bmatrix} 1.25 & 0.75 & 0 \\ 0.75 & 0.25 & 0 \\ 0 & 0 & 5 \end{bmatrix} \tag{17}$$

which indicates the following.

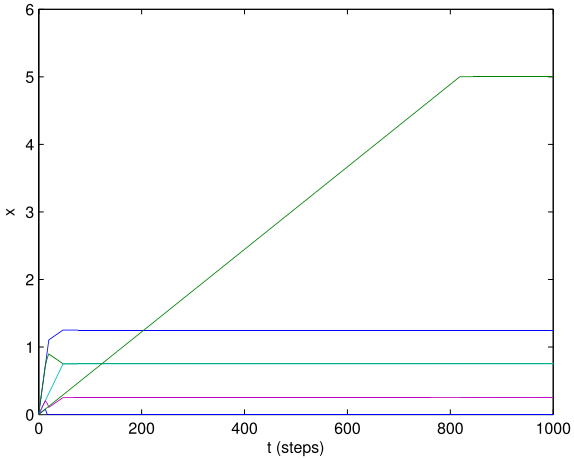
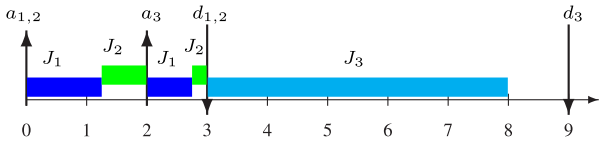


Fig. 3. Transient behaviors of the state variables of RNN (15) in Example 1.

1) Within the first interval $I_1 = [0, 2)$, J_1 needs to executed for 1.25 time units, and J_2 needs to be executed for 0.75 time units.
2) J_1 and J_2 should be executed for 0.75 and 0.25 time units, respectively, during the second interval $I_2 = [2, 3)$.
3) The remaining job J_3 takes over the last interval $I_3 = [3, 9)$, and executes for 5 time units.

This corresponding schedule and execution may be expressed in the following way, where the release time instants are denoted by up arrows, while the deadlines are represented by down arrows, and all allocated amounts are executed in the order of their deadlines within each interval ($J_1 \triangleright J_2 \triangleright J_3$) (please use a colored monitor or printer for better view-in the B&W environment, you may refer to the added job indices above the most execution fragments).



Obviously, this is one of the optimal solutions to the original problem, since all deadlines are met. The less important job J_3 is assigned a lower priority, and does not start to execute until the other two jobs are finished ($t = 3$), while J_1 and J_2 require a total execution of 3 time units within interval [0, 3). For this example, any execution order or swapping within the first two intervals may lead to a feasible schedule (and thus an optimal utility).

Remark: In the case that each job is either fully finished, or not executed at all in the calculated scheduling, i.e., the sum of the ith column of X^* is either c_i or 0, there is a much simpler run-time strategy that would avoid unnecessary preemptions, which is eliminating the jobs with 0 assigned execution, and execute the rest under EDF.

IV. EXPERIMENTS

In this section, based on a large number of randomly generated job sets, we compare the gained utility with typical

TABLE II
PARAMETERS OF JOBS CONSIDERED IN EXAMPLE 2

Job	a_i	d_i	c_i	w_i
1	1	4	2	0.319
2	2	5	1	0.297
3	3	6	2	0.424
4	4	7	1	0.117
5	0	10	6	0.506

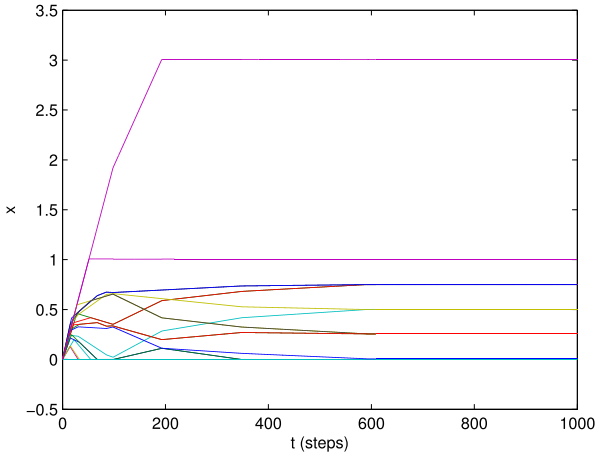


Fig. 4. Transient behaviors of the state variables of RNN (15) in Example 2.

scheduling strategies, and analyze the converging time of the proposed method.

A. Overloaded Example

First, an overloaded set is chosen to further demonstrate our algorithm and to show the necessity of a large-scale experimental comparison.

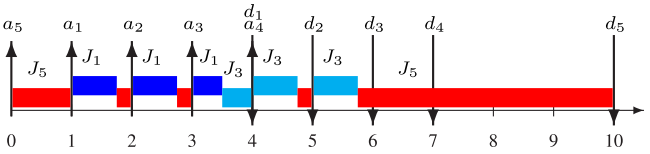
Example 2: We now consider another job set with load greater than 1, and its parameters are described in Table II.

Fig. 4 shows the convergence process of the RNN model. Compared with Fig. 3, we notice that, although the number of jobs is increasing, the performance (convergence time) of our RNN-based method does not suffer any dropping at all. A more thorough and general study is shown in Section IV-C regarding the relationship between the number of steps and the problem size.

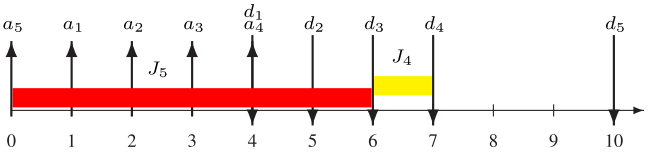
The solution given by the RNN-based approach is transformed into the following scheduling table:

$$X^* = \begin{bmatrix} 0 & 0.75 & 0.75 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.75 & 0.75 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0.25 & 0.25 & 0 & 0.25 & 0.25 & 1 & 3 \end{bmatrix}. \quad (18)$$

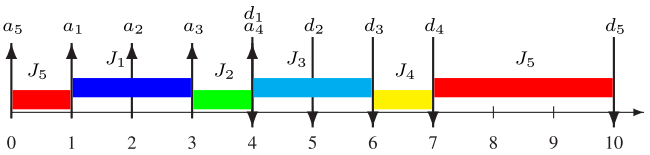
The corresponding execution is shown below (we omit the transformation details for this example), where each job is assigned a different color. Less important jobs (J_2 and J_4) are totally dropped (with no execution). Such execution results in a total utility of $w_1 + w_3 + w_5 = 1.249$, which can be verified (in exponential time) and maximized.



In comparison with our RNN-based method, if we do fixed-priority scheduling (according to the job values w_i), jobs will be executed in the order: $J_5 \rhd J_3 \rhd J_1 \rhd J_2 \rhd J_4$, which results in an execution shown in the following chart. Only two deadlines (d_4 and d_5) are met, which means that the total utility ($w_1 + w_2 = 0.623$) is far from being maximized.



If we apply EDF to the job set, the corresponding executions are shown as below, where J_5 as the most important job will miss its deadline, and the total utility ($w_1 + w_2 + w_3 + w_4 = 1.157$) is not maximized as well.



Unlike Example 1, which is polynomial time solvable, Example 2 provides an overloaded case with much more jobs, which is NP-hard to solve. However, our RNN-based method is still able to find an optimal solution (with maximum utility) in about the same amount of steps as Example 1. We further discuss such optimal behavior in Section V-A based on a theoretical analysis.

B. Comparison Study

In Section IV-A, it has been shown by a specific example that our algorithm outperforms EDF and fixed priority (under the order of weights). Since it is still too early to draw any firm conclusion based upon the given two specific examples, we randomly form 10 000 job sets based on the generator described in [32], where the penalty coefficient w_i is enumerated within the range [0, 1] instead of rounded up as criticality levels.

The benchmark for a direct comparison is a utility ratio-where the gained utility is normalized by the total weight ($\sum_i w_i$). For example, the utility ratios in Example 2 are 0.751, 0.375, 0.751, and 0.696 for the RNN-based method, EDF, the robust-EDF (RED) algorithm [35], and fixed priority, respectively. For fixed priority, we prioritize jobs with larger weights, where jobs with lower values (weights) are dropped under overload conditions. Note that a utility ratio of 1 indicates that all jobs are finished on time, and is impossible under overloaded sets (with load greater than 1).

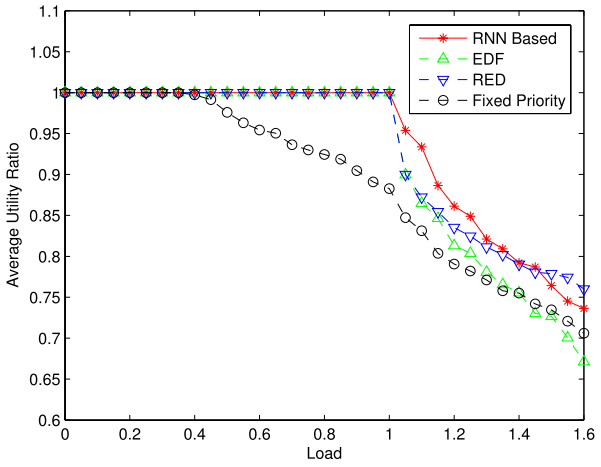


Fig. 5. Utility comparison of the proposed RNN-based method over EDF and fixed priority.

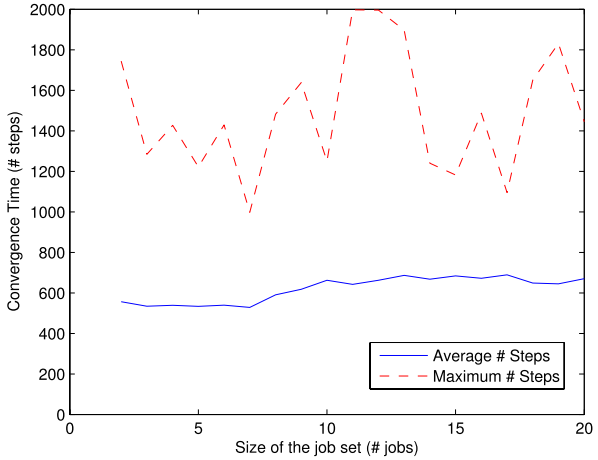


Fig. 6. Average converging time (in number of steps) comparison for different sizes of job sets.

The reason for using the total weight as the denominator is the NP-hardness of finding the exact possible utility. As a result, as certain drop of the utility ratio when load is >1 may not indicate a nonoptimal solution-like in Example 2, no schedule can achieve a utility ratio higher than 0.751, and thus the RNN-based method is already optimal.

All 10 000 generated job sets are categorized by their loads (Definition 3) in a range of 0.05. The average utility ratio in each load group of the discussed three methods are compared in Fig. 5. We can see that the proposed RNN-based method is achieving a much higher utility than EDF and fixed priority on average (here, we do not consider too heavily loaded sets with load greater than 1.6). Under nonoverloaded conditions, both EDF and our algorithm are optimal. However, EDF suffers from a significant drop when the load exceeds 1, and keeps dropping in a faster rate than the other two algorithms while the load increases. On the other hand, fixed priority (in the order of jobs' weights) starts to be nonoptimal very early when the load of the set reaches only $\sim$0.4.

The performance behavior against RED is quite interesting-average speaking, the proposed method outperforms RED when the set is lightly overloaded (load less than 1.4), while the RED results in a higher average utility when the loads of sets become larger. We claim that there is no direct application of RED to our RNN-based framework due to its complicated heuristic, and thus the further possible improvement is left as further work.[3]

C. About Converging Time

In Sections IV-A and IV-B, two examples are generated, where Example 1 contains three jobs and two intervals, and Example 2 contains five jobs and eight intervals. From the experiments, we observe that, although the size of scheduling table (i.e., the number of scheduling decision variables) of Example 2 is about seven times larger, the convergence time does not grow significantly, and even experiences some dropping.

Based on the 10 000 randomly generated sets, we apply the proposed algorithm and record the numbers of steps it takes for the RNN to reach a steady state (where no variable x_i is changing accumulatively larger than 10^{-3} of the absolute value within the last 50 steps). Fig. 6 shows the relationship between the convergence time and the job set size, where the average numbers of steps are shown by the blue solid line, and the maximum needed steps are shown by the red dashed line.

Note that the number of variables (i.e., the size of the scheduling table) is $O(n^2)$, where n is the number of jobs. If the number of converging steps of the introduced RNN model is linearly increasing, we should definitely see a squared line in Fig. 6. We are thus very pleased to observe that the convergence time (number of steps) remains at about the same level when the size of the problem varies. The simulation time for each step to operate is indeed linearly correlated with the number of neurons, and thus is increasing at $O(n^2)$ rate (on our PC). However, due to the natural structure of the adopted RNN system, operations within each step can be easily done paralleled. Moreover, the time for each step to operate remains constant (at a level of 10^{-6} s) when hardware implemented.

Based on our further detailed observation, although the possibility of requiring a relatively large number (>1000) of steps slightly increases (slower than logarithmic speed) when the job set becomes larger, the necessary step ranges for the system to stabilize remain about at the same level for sets with different number of jobs. Since the computer simulation of such RNN system is quite time-consuming (unlike when implemented on real circuits), experiments are not sufficient to draw any firm conclusion on this. Precise studies on the convergence time increasing over the size of job sets are left as further work. Nevertheless, it is safe to claim that the average converging time remains almost at a constant rate when the size of job sets increases.

[3]We would also like to point out that in our experiment we focus more on the average performance, since it has been shown [36] that no method (including ours) can have a competitive factor greater than p in the worst case, where p satisfies $4[1-(\text{load}-1)p]^3 = 27p^2$. Some more existing results for the worst case analysis to overload conditions can be found in [2, Ch. 9].

TABLE III
PARAMETERS OF THE JOBS IN THEOREM 2

Job	a_i	d_i	c_i	w_i
1	0	1	1/2+ϵ/3	1/2+ϵ/2
2	0	1	1/2	1/2
3	0	1	1/2	1/2

V. ANALYSIS AND EXTENSION

A. Optimality Analysis

In Example 2, an overloaded job set is considered, and we apply an RNN based on a piecewise linear (approximated) utility function to solve the original problem with a step objective function. Fortunately, the solution it achieves (by dropping two less important jobs) is also the optimal solution to the NP-hard problem. Here, we try to discuss about such fortunate behaviors, and also give the lower bound performance when we are experiencing the most misfortune.

Although optimizing (6) under constraints (3) and (4) is NP-hard, we are not surprised that, for many job sets, the proposed approximate RNN approach performs quite well, and is very likely to converge to the actual optimal solution of the scheduling problem.

The reason for this is that, even overloaded, only a slight number of them may need to be skipped. Thus, in many cases, it is correct to greedily select the less important jobs for dropping. Our RNN-based design is only changing part of the objective function, and gives linear penalty according to the unfinished length after deadline. This change does not affect the objective for finishing a job on time.

Actually, in many real-time scheduling applications, such as safety-critical systems, people do not really care about whether this NP-hard problem is perfectly solved-as far as the critical (more important) jobs are finished on time, any reasonable choice upon the rest (less important) jobs in overloaded cases is acceptable. Regarding our method, by setting the importance value (the slope of the utility function) large enough, the penalty for not finishing the critical job on time significantly increases, and thus its on-time finishing will be guaranteed.

To provide a worst case boundary to the algorithm, an approximation ratio of 0.5 will be derived. Note that, for the overloaded case in online scheduling, where the release time of the jobs remains unknown until they are active, the approximation ratio has been well studied [36]–[38]. Here, we are facing a different problem, where the release time is known *a priori*.

Theorem 2: Our algorithm has a worst approximation ratio[4] of at most 0.5 under the overloaded case.

Proof: To prove the theorem, all we need to show is that there exists an overloaded job set, on which if our algorithm gains a total utility of x, other schedule may achieve a utility any close (but smaller than) $2x$.

Consider the following set shown in Table III, where $\epsilon > 0$ is a small positive constant.

Our RNN-based approach will try to finish J_1 first, and results in a total utility of $1/2 + \epsilon/2$, while a best effort would finish both J_2 and J_3, which leads to a utility of 1. ■

Theoretically speaking, each job may reach its WCET. However, practically speaking, most of the time jobs will finish their executions quite beforehand. If we take another look at the job set in Table III, executing them in the order of what the RNN method suggests has a close to 1 probability of finishing both J_1 and J_2, which leads to a larger total utility of $1 + \epsilon/2$. Although scheduling problems can be equivalently transformed into the NP-hard knapsack problem, unlike knapsack where jobs are always considered as a whole, and it is either selected or not (0/1), in many practical cases, executing a job for a portion of its WCET is much better than dropping it at the beginning-it already gave the job a certain (and maybe large enough) chance to finish. A more complicated schedulability analysis regarding probability WCET models can be done (see [39], [40]), but is beyond the scope of this paper.

B. Extension

1) About Response Time Minimization: In some nonoverloaded cases, although it is *a priori* known that there exist some feasible schedules (e.g., EDF) upon which all deadlines will be met, we still wish to have some specific jobs to be finished as early as possible. That is to say, upon finishing all jobs on time, reward can be further gained by swapping some pieces of executions to make certain jobs finish as early as possible.

We introduce slack variables y_i, denoting the finishing time instants for each job J_i under a given schedule table X. The following two sets of linear inequality constraints well define and restrict the introduced slack variable y_i.

First, constraints need to be added specifying that each job receives adequate execution upon this finishing interval

$$\forall i, \sum_{k|a_i \le t_k < y_i} x_{i,k} \ge c_i. \tag{19}$$

Although the finishing interval I_k can be derived, the actual finishing time may lie anywhere between t_k and t_{k+1}. Therefore, second, we assume that jobs are executed in the EDF order within each interval according to the scheduling table.[5] Thus, the exact finishing time y_i of a given job J_i can be derived from the allocated amounts within the corresponding interval I_k

$$\forall i, \sum_{j|j \le i} x_{j,k} \le y_i - t_k, \quad \text{for } k \text{ s.t. } t_k < y_i \le t_{k+1}. \tag{20}$$

Given the finishing time for each job, the average response time can be minimized through

$$\max_{X,y} \sum_{i=1}^{n} \varphi(y_i). \tag{21}$$

[4]The approximation ratio is a common evaluation metric for polynomial-time algorithms for NP-hard problems, which has nothing to do with the previously mentioned utility ratio in the experiments.

[5]Other execution priority orders within each interval, such as most important job first, may also serve as reasonable choices, with a different set of constraints to be derived.

The piecewise linear function φ may be chosen as the ones shown in Fig. 2(a) and (b), depending on whether the scheduler cares about deadline meeting itself, or the response time to a given job as well. Since all deadlines can be met, feasible solutions will lie on the left half of each utility function [to the left of d_i in Fig. 2(a) and (b)]. Thus, only one of the slope parameters (l_i) is affecting the searching process of our dynamic system. Different values of l_i may be chosen to differentiate the importance of finishing a particular job as early as possible, and a zero slope suggests that, as far as the deadline is met, it is meaningless to finish the job any earlier.

2) About Soft Deadlines and Tardiness Minimization: In soft real-time scheduling, deadlines are no longer hard constraints, and not only the executions before deadline matter but also the ones afterward need to be considered. Since there may be execution after the last deadline $t_k = \max_i\{d_i\}$, we need to introduce one more interval $I_{k+1} = [t_k, +\infty)$ to the system, which also results in an additional column to the scheduling table X.

In order to form the objective in a similar way (by applying the inverse step function), the slack variables y_i need to be applied as well with constraints (19), (20), and the objective for the tardiness minimization in soft real-time scheduling can be expressed in the same way as (21).

In tardiness minimization problems, for each job J_i, a certain value w_i is added to the total utility as far as the deadline d_i is met, while even when d_i is missed, earlier finishing the job will gain a (linear correspondingly) larger portion of its value. Mathematically speaking, the piecewise linear function φ may be similar to the one shown in Fig. 2(a). Unlike the previous (response time minimization) case, here, jobs may finish before, at, or after their deadlines, and thus y_i may lie anywhere after a_i in Fig. 2(a). The slope parameter l_i will be set to zero, since it has nothing to do with tardiness (corresponding to cases when jobs are finished before their deadlines). As a result, only the slope parameter h_i will affect the searching process of our dynamic system, and various values of h_i may be chosen to differentiate the importance of having a particular job's tardiness as small as possible. While a negative infinity slope suggests that it is a hard deadline-as far as the deadline is missed, the utility of the whole system goes to negative infinity immediately.

For the above mentioned two scheduling concerns, various choices of parameters of the utility function may lead to different schedules-comparisons can be made only when specific QoS definitions are given. Since the purpose of this paper is to bring RNNs into the game (of real-time scheduling), we will not jump into any messy details by explaining scheduling application examples. The discussions in this section are only trying to convince the reader that the piecewise linear utility function model we introduced is very powerful, into which many interesting real-time scheduling problems may be transformed.

VI. Conclusion

Many novel RNN models have recently been proposed for solving optimization problems with linear inequality constraints. These RNN models are often with very simple structures, and converge to the global optima rapidly. Due to the parallel nature structure of the RNNs, these models may be applied on parallel computing devices. Moreover, the RNN-based approaches have the potential of being implemented on hardware. As a result, the converging periods (into a stable state) of such systems can be extremely short compared with the execution length of real-time jobs.

To investigate the potential of applying RNNs in real-time scheduling, in this paper, we apply one of the RNN models on a series of real-time scheduling problems. We have presented rules for transformation and approximation from some typical NP-hard real-time scheduling problems into RNN solvable problems, and shown how they work out by examples. Experimental studies suggest that the convergence time of the introduced neurodynamic system is likely to stay in a constant range when the size of job set grows, which indicates that our method may serve as a scheduler for large-scale platforms, e.g., super computers, computing grids, and cloud centers.

Based on the randomly generated 10 000 job sets, comparison studies have been reported. It is evident that the proposed RNN-based method outperforms EDF and fixed priority under overloaded conditions, while remains optimal (same as EDF) in nonoverloaded conditions. Compared with any possible optimal scheduler (NP-hard problem solver), an approximation ratio bound of 0.5 is demonstrated by a designed example. Here, 0.5 is only an upper bound of the ratio-a tight one has not been derived yet.

Various functions can be chosen to approximate the step objective function (where NP-hard arises from) in scheduling-related optimization problems. All the discussion and analysis in this paper are based on one of the choices, which is using the concave piecewise linear function (polynomial time solvable). Some other functions, e.g., sigmoid, may be applied for better approximation (in the shape), while only converging to suboptimal. To guarantee a larger probability of converging to optima, parameter tuning may be critical, and requires a careful experimental study.

We only target the scheduling problem on independent one shot job set on a uniprocessor platform. A larger portion of research in real-time scheduling focuses on multicore systems and the periodic/sporadic task set model, and may serve as a promising further work direction. Besides, preemptions are assumed at no cost in this paper, and thus our RNN model may introduce unnecessary preemptions during its search (e.g., Example 2). An additional term in the utility function regarding preemption numbers is worth investigating as well.

References

[1] J. W. S. W. Liu, *Real-Time Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, 2000.

[2] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, vol. 24. New York, NY, USA: Springer-Verlag, 2011.

[3] U. Devi, "Soft real-time scheduling on multiprocessors," Ph.D. dissertation, Dept. Comput. Sci., Univ. North Carolina Chapel Hill, Chapel Hill, NC, USA, 2006.

[4] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[5] B. Ravindran, E. D. Jensen, and P. Li, "On recent advances in time/utility function real-time scheduling and resource management," in *Proc. 8th IEEE Int. Symp. Object-Oriented Real-Time Distrib. Comput. (ISORC)*, May 2005, pp. 55–60.
[6] A.-H. Mohsenian-Rad, V. W. S. Wong, J. Jatskevich, R. Schober, and A. Leon-Garcia, "Autonomous demand-side management based on game-theoretic energy consumption scheduling for the future smart grid," *IEEE Trans. Smart Grid*, vol. 1, no. 3, pp. 320–331, Dec. 2010.
[7] H. Wu, B. Ravindran, E. D. Jensen, and P. Li, "Energy-efficient, utility accrual scheduling under resource constraints for mobile embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 5, no. 3, pp. 513–542, 2006.
[8] A. Burns and R. I. Davis. (2013). *Mixed-Criticality Systems: A Review*. [Online]. Available: http://www-users.cs.york.ac.uk/~burns/review.pdf
[9] D. Tank and J. J. Hopfield, "Simple 'neural' optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit," *IEEE Trans. Circuits Syst.*, vol. 33, no. 5, pp. 533–541, May 1986.
[10] S. Zhang and A. G. Constantinides, "Lagrange programming neural networks," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 39, no. 7, pp. 441–452, Jul. 1992.
[11] J. Wang, "A deterministic annealing neural network for convex programming," *Neural Netw.*, vol. 7, no. 4, pp. 629–641, 1994.
[12] Y. Xia, "A new neural network for solving linear and quadratic programming problems," *IEEE Trans. Neural Netw.*, vol. 7, no. 6, pp. 1544–1548, Nov. 1996.
[13] M. Forti, P. Nistri, and M. Quincampoix, "Generalized neural network for nonsmooth nonlinear programming problems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 9, pp. 1741–1754, Sep. 2004.
[14] X. Hu and J. Wang, "Solving pseudomonotone variational inequalities and pseudoconvex optimization problems using the projection neural network," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1487–1499, Nov. 2006.
[15] W. Bian and X. Xue, "Subgradient-based neural networks for nonsmooth nonconvex optimization problems," *IEEE Trans. Neural Netw.*, vol. 20, no. 6, pp. 1024–1038, Jun. 2009.
[16] Z. Guo, Q. Liu, and J. Wang, "A one-layer recurrent neural network for pseudoconvex optimization subject to linear equality constraints," *IEEE Trans. Neural Netw.*, vol. 22, no. 12, pp. 1892–1900, Dec. 2011.
[17] Z. Zeng and T. Huang, "New passivity analysis of continuous-time recurrent neural networks with multiple discrete delays," *J. Ind. Manage. Optim.*, vol. 7, no. 2, pp. 283–289, 2011.
[18] L. Cheng, Z.-G. Hou, Y. Lin, M. Tan, W. C. Zhang, and F.-X. Wu, "Recurrent neural network for non-smooth convex optimization problems with application to the identification of genetic regulatory networks," *IEEE Trans. Neural Netw.*, vol. 21, no. 5, pp. 714–726, May 2011.
[19] Q. Liu, Z. Guo, and J. Wang, "A one-layer recurrent neural network for constrained pseudoconvex optimization and its application for dynamic portfolio optimization," *Neural Netw.*, vol. 26, pp. 99–109, Feb. 2012.
[20] C. Guo and Q. Yang, "A neurodynamic optimization method for recovery of compressive sensed signals with globally converged solution approximating to ℓ_0 minimization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 7, pp. 1363–1374, Jul. 2015.
[21] Q. Liu and J. Wang, "Finite-time convergent recurrent neural network with a hard-limiting activation function for constrained optimization with piecewise-linear objective functions," *IEEE Trans. Neural Netw.*, vol. 22, no. 4, pp. 601–613, Apr. 2011.
[22] J. Y.-T. Leung, Ed., *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Boca Raton, FL, USA: CRC Press, 2004.
[23] C. Cardeira and Z. Mammeri, "Neural networks for multiprocessor real-time scheduling," in *Proc. IEEE 6th Euromicro Workshop Real-Time Syst.*, Jun. 1994, pp. 59–64.
[24] C. Cardeira and Z. Mammeri, "Neural network versus max-flow algorithms for multiprocessor real-time scheduling," in *Proc. IEEE 8th Euromicro Workshop Real-Time Syst.*, Jun. 1996, pp. 175–180.
[25] M. P. Silva, C. Cardeira, and Z. Mammeri, "Solving real-time scheduling problems with Hopfield-type neural networks," in *Proc. 23rd EUROMICRO Conf. New Frontiers Inf. Technol.*, Sep. 1997, pp. 671–678.
[26] Y.-M. Huang and R.-M. Chen, "Scheduling multiprocessor job with resource and timing constraints using neural networks," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 29, no. 4, pp. 490–502, Aug. 1999.
[27] R.-M. Chen and Y.-M. Huang, "Competitive neural network to solve scheduling problems," *Neurocomputing*, vol. 37, nos. 1–4, pp. 177–196, 2001.
[28] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.
[29] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. 28th IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2007, pp. 239–243.
[30] H. Leontyev, "Compositional analysis techniques for multiprocessor soft real-time scheduling," Ph.D. dissertation, Dept. Comput. Sci., Univ. North Carolina Chapel Hill, Chapel Hill, NC, USA, 2010.
[31] J. P. Erickson, "Managing tardiness bounds and overload in soft real-time systems," Ph.D. dissertation, Dept. Comput. Sci., Univ. North Carolina Chapel Hill, Chapel Hill, NC, USA, 2014.
[32] S. K. Baruah and Z. Guo, "Mixed-criticality scheduling upon varying-speed processors," in *Proc. IEEE 34th Real-Time Syst. Symp. (RTSS)*, Dec. 2013, pp. 68–77.
[33] Z. Guo and S. K. Baruah, "Implementing mixed-criticality systems upon a preemptive varying-speed processor," *Leibniz Trans. Embedded Syst.*, vol. 1, no. 2, pp. 3:1–3:19, 2014.
[34] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Berlin, Germany: Springer-Verlag, 2004.
[35] G. C. Buttazzo and J. A. Stankovic, "RED: Robust earliest deadline scheduling," in *Proc. 3rd Int. Workshop Responsive Comput. Syst.*, 1993, pp. 100–111.
[36] S. Baruah *et al.*, "On the competitiveness of on-line real-time task scheduling," in *Proc. 12th IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 1991, pp. 106–115.
[37] G. Koren and D. Shasha, "D^{over}: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems," *SIAM J. Comput.*, vol. 24, no. 2, pp. 318–339, Apr. 1995.
[38] B. Kalyanasundaram and K. Pruhs, "Speed is as powerful as clairvoyance," *J. ACM*, vol. 47, no. 4, pp. 617–643, 2000.
[39] D. Maxim, O. Buffet, L. Santinelli, L. Cucu-Grosjean, and R. I. Davis, "Optimal priority assignment algorithms for probabilistic real-time systems," in *Proc. 19th Int. Conf. Real-Time Netw. Syst. (RTNS)*, 2011, pp. 129–138.
[40] Z. Guo, L. Santinalli, and K. Yang, "EDF schedulability analysis on mixed-criticality systems with permitted failure probability," in *Proc. 21st IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2015, pp. 187–196.

Zhishan Guo (S'10) received the B.E. degree in computer science and technology from Tsinghua University, Beijing, China, in 2009, and the M.Phil. degree in mechanical and automation engineering from the Chinese University of Hong Kong, Hong Kong, in 2011. He is currently pursuing the Ph.D. degree with the Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA.

His current research interests include real-time scheduling, cyber-physical systems, and neural networks and their applications.

Sanjoy K. Baruah (F'13) received the Ph.D. degree from the University of Texas at Austin, Austin, TX, USA, in 1993.

He is currently a Professor with the Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA. His current research interests include real-time and safety-critical system design, and resource-allocation and sharing in distributed computing environments.