Automated Competitive Analysis of Real-time Scheduling with Graph Games

Krishnendu Chatterjee
IST Austria (Institute of Science and Technology Austria)
Maria Gugging, Austria
Email: krish.chat@ist.ac.at

Alexander Kößler and Ulrich Schmid
Embedded Computing Systems Group (E182/2)
Vienna University of Technology
Vienna, Austria
Email: {koe,s}@ecs.tuwien.ac.at

***Abstract*—In this paper,[1] we introduce the powerful tool of algorithmic game theory for competitive analysis of real-time scheduling with firm deadlines. We introduce a novel instance of a partial-observation game that is suitable for this purpose, and prove that all involved decision problems are solveable. Apart from deriving a graph game that allows the automated computation of the competitive ratio, along with an NP-completeness proof of the decision problems involved, our approach also admits polynomial solutions for computing both the worst-case utility and the worst-case utility ratio w.r.t. a clairvoyant off-line algorithm, for a given on-line algorithm. A particularly interesting feature of the proposed approach lies in the fact that additional constraints on the adversary and/or on the algorithm, including limited maximum or average load, finiteness of periods of overload, etc. are easily added.**

I. Introduction

Competitive analysis [5] allows to compare the performance of an on-line algorithm $\mathcal{A}$, processing a sequence of inputs without knowing the future, with what can be achieved by an optimal off-line algorithm C that does know the future (a clairvoyant algorithm). Its primary focus is the *competitive ratio*, which gives the worst-case performance ratio of $\mathcal{A}$ vs. C taken over all input scenarios.

This paper is devoted to the competitive analysis of deterministic real-time scheduling algorithms. Their purpose is to schedule a sequence of dynamically arriving tasks, which have specific execution time requirements and firm deadlines, on a single processor: A task that is completed by its deadline contributes some positive utility value to the system; a task that does not make its deadline does not harm but does not add any utility. The goal of the scheduling algorithm is to maximize the cumulated utility.

In their seminal paper [2], Baruah et. al. proved that no on-line algorithm can achieve a competitive ratio better than 1/4 over a clairvoyant algorithm. The proof is based on constructing a specific task sequence, which takes into account the on-line algorithm's actions and thereby forces it to deliver a sub-optimal cumulated utility. By also giving an on-line algorithm TD1 with competitive ratio 1/4, they prove that 1/4 is indeed the best competitive ratio one can achieve for this real-time scheduling problem.

[1]This work has been supported by the Austrian Science Foundation (FWF) under the NFN RiSE (S11405), NFN RiSE (S11407), FWF Grant No P 23499-N23, ERC Start grant (279307: Graph Games), and Microsoft faculty fellows award.

Obviously, the above situation can be viewed as an instance of a game between the on-line algorithm (Player 1) and an adversary (Player 2) that determines the task sequence. The question addressed in this paper is whether and how the powerful "tool" of the *theory of games on graphs* [13], [16] can be utilized for competitive analysis of real-time scheduling algorithms. To the best of our knowledge, this question has not been addressed in literature before. We can answer it in the affirmative, at least from a theoretical perspective; from a practical point of view, future research is required in order to address time complexity issues.

Our paper makes the following contributions:

(1) In Sect. II, we introduce a game model (a partial-observation game with mean-payoff and ratio objectives) suitable for competitive analysis of real-time scheduling algorithms. The mean-payoff resp. ratio objective allows to compute the cumulated utility resp. the competitive ratio of the best on-line algorithm under the worst-case task sequence.
(2) In Sect. III, we prove that the relevant decision problems, both for mean-payoff and ratio objectives, are decideable. For our general partial-information game setting, we show that they are NP-complete in the size of the underlying game graph.
(3) In Sect. IV, we provide a finite-state *labeled transition system* (LTS) representation of the real-time scheduling algorithms of interest, which naturally leads to the corresponding game graphs. A suitable reward function is used for expressing utility values.
(4) In Sect. V-A, we show that worst-case average cumulated utitities can be derived from a perfect-information game variant of the above model. Consequently, deciding whether there is an on-line algorithm with worst-case average cumulated utility above some threshold is in $\text{NP} \cap \text{coNP}$ in general, and polynomial in the size of the game graph for reasonable choices of task utility values. Automatically determining the optimal values of the worst-case average cumulated utility, as well as constructing an optimal real-time scheduling algorithm achieving it, can be done efficiently as well. A polynomial solution also exists for computing the worst-case utility ratio, which relates the worst-case cumulated

utility of a given on-line algorithm to the worst-case cumulated utility of a clairvoyant off-line algorithm.

(5) In Sect. V-B, we show that the general partial-information game model is suitable for dealing with the ratio objective, i.e., is suitable for competitive analysis. It follows that deciding whether there is an on-line algorithm with competitive ratio above some threshold is in NP w.r.t. the size of the underlying game graph.

(6) In Sect. VI, we show that additional constraints on the adversary, like limited maximum load, finite duration of overload, as well as limited averages of additional constraints, can easily be accommodated: In fact, any safety, Büchi and multiple mean-payoff objective can be added without unduly increasing the complexity of the decision problems involved.

Some conclusions and directions of future work in Sect. VII round off our paper.

II. Partial-observation Mean-payoff and Ratio Games

In this section we will first present the basic definitions of partial-observation games with mean-payoff and ratio objectives, and then present the complexity results.

A. Basic definitions

We start with the basic definitions of partial-observation games and their subclasses, and the notions of strategies and objectives. We focus on partial-observation games, where one player has partial observation and the other player has perfect observation.

Partial-observation games. A *partial-observation game* (or simply a *game*) is a tuple $\mathsf{G} = \langle S^G, \Sigma_1^G, \Sigma_2^G, \delta^G, \mathcal{O}_S, \mathcal{O}_\Sigma \rangle$ with the following components:

1) *(State space).* The set S^G is a finite set of states.
2) *(Actions).* Σ_i^G $(i = 1, 2)$ is a finite set of actions for Player i.
3) *(Transition function).* The transition function $\delta^G : S^G \times \Sigma_1^G \times \Sigma_2^G \to S^G$ given the current state $s \in S^G$, an action $\sigma_1 \in \Sigma_1^G$ for Player 1, and an action $\sigma_2 \in \Sigma_2^G$ for Player 2, gives the next (or successor) state $s' = \delta^G(s, \sigma_1, \sigma_2)$.
4) *(Observations).* The set $\mathcal{O}_S \subseteq 2^S$ is a finite set of observations for Player 1 that partitions the state space S^G. The partition uniquely defines function $\mathsf{obs}_S : S^G \to \mathcal{O}_S$ that map each state to its observation such that $s \in \mathsf{obs}_S(s)$ for all $s \in S^G$. In other words, the observation partitions the state space according to equivalence classes. Similarly, $\mathcal{O}_\Sigma$ is a finite set of observations for Player 1 that partitions the action set Σ_2^G, and analogously define the function obs_Σ. Intuitively, Player 1 will have partial observation, and can only observe the current observation of the state (not the precise state but only the equivalence class the state belongs to) and current observation of the action of Player 2 (but not the precise action of Player 2) to make her choice of action.

Perfect-information games. *Games of complete-observation* (or perfect-information games) are a special case of partial-observation games where $\mathcal{O}_S = \{\{s\} \mid s \in S^G\}$ and $\mathcal{O}_\Sigma = \{\{\sigma_2\} \mid \sigma_2 \in \Sigma_2^G\}$, i.e., every state and action is visible to Player 1, and thus she has perfect information. For perfect-information games, for the sake of simplicity, we will omit the corresponding observation sets from the description of the game.

Plays. In a game, in each turn, first Player 2 chooses an action, then Player 1 chooses an action, and given the current state and the joint actions, we obtain the next state following the transition function δ^G. A *play* in G is an infinite sequence of states and actions $\rho = s^0\sigma_2^0\sigma_1^0 s^1 \sigma_2^1\sigma_1^1 s^2 \sigma_2^2\sigma_1^2 s^3 \dots$ such that, for all $j \geq 0$, we have $\delta^G(s^j, \sigma_1^j, \sigma_2^j) = s^{j+1}$. The *prefix up to* s^n of the play ρ is denoted by $\rho(n)$, its *length* is $|\rho(n)| = n+1$ and its *last element* is $\mathsf{Last}(\rho(n)) = s^n$. The set of plays in G is denoted by $\mathsf{Plays}(\mathsf{G})$, and the set of corresponding finite prefixes is denoted $\mathsf{Prefs}(\mathsf{G})$.

Strategies. A *strategy* for a player is a recipe that specifies how to extend finite prefixes of plays. We will consider *memoryless* (or positional) strategies for Player 1 and general history-dependent strategies for Player 2. A strategy for Player 1 is a function $\pi : \mathcal{O}_S \times \mathcal{O}_\Sigma \to \Sigma_1^G$ that given the current observation of the state and the current observation on the action of Player 2, selects the next action. A strategy for Player 2 is a function $\sigma : \mathsf{Prefs}(\mathsf{G}) \to \Sigma_2^G$ that given the current prefix of the play chooses an action. Observe that the strategies for Player 1 are both observation-based and memoryless; i.e., depend only on the observations and also only on the current observation and action (rather than the whole history), whereas the strategies for Player 2 depend on the history. A memoryless strategy for Player 2 only depends on the last state of a prefix. We denote by Π_{G}^M, Σ_{G}, Σ_{G}^M the set of all observation-based memoryless Player 1 strategies, the set of all Player 2 strategies, and the set of all memoryless Player 2 strategies, respectively. In sequel when we write strategy for Player 1 we consider only observation-based memoryless strategies. Given a strategy π and a stratregy σ for Player 1 and Player 2, and an initial state s^0, we obtain a unique play $\rho(s^0, \pi, \sigma) = s^0\sigma_2^0\sigma_1^0 s^1 \sigma_2^1\sigma_1^1 s^2 \sigma_2^2\sigma_1^2 s^3 \dots$ such that, for all $n \geq 0$, we have $\sigma(\rho(n)) = \sigma_2^n$ and $\pi(\mathsf{obs}_S(s^n), \mathsf{obs}_\Sigma(\sigma_2^n)) = \sigma_1^n$.

Objectives. In this work we will consider *mean-payoff* (or long-run average or limit-average) objectives, as well as *ratio* objectives. For mean-payoff objectives we will consider games with a reward function $r : S^G \times \Sigma_1^G \times \Sigma_2^G \times S^G \to \mathbb{N}$ that maps every transition to a non-negative integer valued reward. In case of ratio objectives, we will consider games with two such reward functions $r_1 : S^G \times \Sigma_1^G \times \Sigma_2^G \times S^G \to \mathbb{N}$ and $r_2 : S^G \times \Sigma_1^G \times \Sigma_2^G \times S^G \to \mathbb{N}$. Given a reward function r and a play $\rho = s^0\sigma_2^0\sigma_1^0 s^1 \sigma_2^1\sigma_1^1 s^2 \sigma_2^2\sigma_1^2 s^3 \dots$, for $n \geq 0$, let $\mathsf{Sum}(r, \rho(n)) = \sum_{i=0}^{n-1} r(s^i, \sigma_1^i, \sigma_2^i, s^{i+1})$ denote the sum of the rewards for the prefix $\rho(n)$. We define the mean-payoff and ratio objectives as follows.

1) The mean-payoff-sup (resp. mean-payoff-inf) objective $\mathsf{MPsup}(r) : \mathsf{Plays}(\mathsf{G}) \to \mathbb{R}$ (resp. $\mathsf{MPinf}(r)$:

$\mathsf{Plays}(\mathsf{G}) \to \mathbb{R}$) for a reward function r, assigns a real-valued reward to every play as follows: for a play ρ we have

$$\mathsf{MPsup}(r)(\rho) = \limsup_{n\to\infty} \frac{1}{n} \cdot \mathsf{Sum}(r, \rho(n));$$

$$\mathsf{MPinf}(r)(\rho) = \liminf_{n\to\infty} \frac{1}{n} \cdot \mathsf{Sum}(r, \rho(n)).$$

2) The ratio objective $\mathsf{Ratio}(r_1, r_2) : \mathsf{Plays}(\mathsf{G}) \to \mathbb{R}$ for two reward functions r_1 and r_2 also assigns a real-value reward to every play as follows: for a play ρ we have

$$\mathsf{Ratio}(r_1, r_2)(\rho) = \liminf_{n\to\infty} \frac{1 + \mathsf{Sum}(r_1, \rho(n))}{1 + \mathsf{Sum}(r_2, \rho(n))}.$$

Note that adding 1 in both numerator and denominator of the ratio avoids division by 0 issues, yet is irrelevant for sums that go to infinity for $n \to \infty$.

Decision problems. The relevant decision problems that we will be interested in are as follows: given a game G, a starting state s^0, reward functions r, r_1, r_2, and a rational threshold $\nu \in \mathbb{Q}$, whether $\sup_{\pi\in\Pi^M_{\mathsf{G}}} \inf_{\sigma\in\Sigma_{\mathsf{G}}} \mathsf{MPinf}(r)(\rho(s^0, \pi, \sigma)) \geq \nu$; and $\sup_{\pi\in\Pi^M_{\mathsf{G}}} \inf_{\sigma\in\Sigma_{\mathsf{G}}} \mathsf{Ratio}(r_1, r_2)(\rho(s^0, \pi, \sigma)) \geq \nu$.

B. Perfect information games

We now present the fundamental results about *perfect-information* games with mean-payoff objectives that follow from the classical results of [9], [18], [10], [6].

Theorem 1: (Determinacy and complexity of perfect-information mean-payoff games [9], [18], [10], [6]). The following assertions hold for perfect-information games with reward function r:

1) *(Determinacy).* We have

$$\begin{aligned}&\sup_{\pi\in\Pi^M_{\mathsf{G}}} \inf_{\sigma\in\Sigma_{\mathsf{G}}} \mathsf{MPinf}(r)(\rho(s^0, \pi, \sigma))\\ &= \inf_{\sigma\in\Sigma_{\mathsf{G}}} \sup_{\pi\in\Pi^M_{\mathsf{G}}} \mathsf{MPsup}(r)(\rho(s^0, \pi, \sigma))\\ &= \inf_{\sigma\in\Sigma^M_{\mathsf{G}}} \sup_{\pi\in\Pi^M_{\mathsf{G}}} \mathsf{MPsup}(r)(\rho(s^0, \pi, \sigma)).\end{aligned}$$

2) Whether $\sup_{\pi\in\Pi^M_{\mathsf{G}}} \inf_{\sigma\in\Sigma_{\mathsf{G}}} \mathsf{MPinf}(r)(\rho(s^0, \pi, \sigma)) \geq \nu$ can be decided in NP$\cap$ coNP, for a rational threshold ν.
3) The computation of the optimal value $v^* = \sup_{\pi\in\Pi^M_{\mathsf{G}}} \inf_{\sigma\in\Sigma_{\mathsf{G}}} \mathsf{MPinf}(r)(\rho(s^0, \pi, \sigma))$ and an optimal memoryless strategy $\pi^* \in \Pi^M_{\mathsf{G}}$ such that $v^* = \inf_{\sigma\in\Sigma_{\mathsf{G}}} \mathsf{MPinf}(r)(\rho(s^0, \pi^*, \sigma))$ can be done in time $O(n \cdot m \cdot W)$, where n is the number of states, m is the number of transitions, and W is the maximal value of all the rewards (i.e., the algorithm is pseudo-polynomial time, and if the maximal value W of rewards is polynomial in the size of the game, then the algorithm is polynomial).
4) For any given memoryless strategy $\pi \in \Pi^M_{\mathsf{G}}$, computing $\inf_{\sigma\in\Sigma_{\mathsf{G}}} \mathsf{MPinf}(r)(\rho(s^0, \pi, \sigma))$ can be done in time $O(n \cdot m \cdot \log W)$.

III. Complexity of General Decision Problems

In this section, we establish the complexity of the decision problems we consider for partial-observation mean-payoff and ratio objectives. In particular, we will establish that for partial-observation games with memoryless strategies for Player 1 all the decision problems are NP-complete.

Transformation. We start with a simple transformation that will allow us to simplify the technical results we prove. In the definition of games, we defined them in such a way that in every state every action was available for the players for simplicity. For technical development, we will consider games where, at certain states, some actions are not allowed for a player. The transformation of such games to games where all actions are allowed is as follows: we add two dummy states (with only a self-loop), one for Player 1 and the other for Player 2, and assign them rewards in a way such that the objectives are violated for the player. For example, for mean-payoff objectives with threshold $\nu > 0$, we assign reward 0 for the Player 1 dummy state, and a reward strictly greater than ν for the Player 2 dummy state; and in case of ratio-objectives we assign the reward pairs similarly. Given a state s, if Player 1 plays an action that is not allowed at s, we goto the dummy Player 1 state; and if Player 2 plays an action that is not allowed we go to the Player 2 dummy state. Thus we have a simple linear time transformation. Hence, for technical convenience, we can assume in the sequel that different states have different set of available actions for the players. We first start with the hardness result.

Lemma 1: The decision problems for partial observation game mean-payoff and ratio objectives, i.e, whether $\sup_{\pi\in\Pi^M_{\mathsf{G}}} \inf_{\sigma\in\Sigma_{\mathsf{G}}} \mathsf{MPinf}(r)(\rho(s^0, \pi, \sigma)) \geq \nu$ (resp. $\sup_{\pi\in\Pi^M_{\mathsf{G}}} \inf_{\sigma\in\Sigma_{\mathsf{G}}} \mathsf{Ratio}(r_1, r_2)(\rho(s^0, \pi, \sigma)) \geq \nu$), are NP-hard in the strong sense.

Proof: We present a reduction from the 3-SAT problem, which is NP-hard in the strong sense [15]. Let Φ be a 3-SAT formula over n variables $x_1, x_2, \ldots, x_n$ in conjunctive normal form, with m clauses $C_1, C_2, \ldots, C_m$ consisting of a disjunction of 3 literals (a variable x_k or its negation $\overline{x_k}$) each. We will construct a game graph G_Φ as follows:

1) *(State space).* $S^G = \{s_{\mathsf{init}}\} \cup \{s_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq 3\} \cup \{\mathsf{dead}\}$; i.e., there is an initial state s_{init}, a dead state dead, and there is a state $s_{i,j}$ for every clause i and a literal j of i.
2) *(Actions).* The set of actions for Player 1 is $\{\mathsf{true}, \mathsf{false}, \bot\}$ and for Player 2 is $\{1, 2, \ldots, m\} \cup \{\bot\}$.
3) *(Transition).* In the initial state s_{init}, Player 1 has only one action $\bot$ available, and Player 2 has actions $\{1, 2, \ldots, m\}$ available, and given action $1 \leq i \leq m$, the next state is $s_{i,1}$. In all other states Player 2 has only one action $\bot$ available. In states $s_{i,j}$ Player 1 has two actions available, namely, true and false. The transitions are as follows:
   - If the action of Player 1 is true in $s_{i,j}$, then (i) if the j-th literal in C_i is x_k, then we have a transition back to the initial state; and (ii) if the j-th literal in

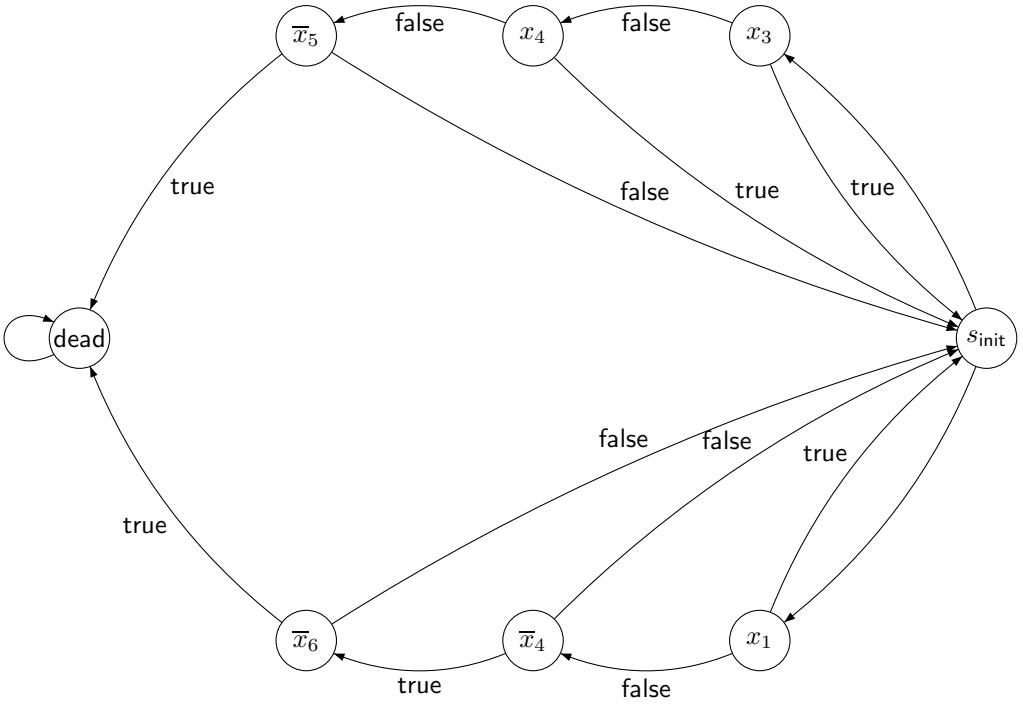

Fig. 1. Illustration of construction of the game from a formula.

C_i is $\overline{x}_k$ (negation of x_k), then we have a transition to $s_{i,j+1}$ if $j \in \{1,2\}$, and if $j = 3$, we have a transition to dead.

- If the action of Player 1 is false in $s_{i,j}$, then (i) if the j-th literal in C_i is $\overline{x}_k$ (negation of x_k), then we have a transition back to the initial state; and (ii) if the j-th literal in C_i is x_k, then we have a transition to $s_{i,j+1}$ if $j \in \{1,2\}$, and if $j = 3$, we have a transition to dead.

In state dead both players have only one available action $\bot$, and dead is a state with only a self-loop (transition only to itself).

4) *(Observations).* First, Player 1 does not observe the actions of Player 2 (i.e., Player 1 does not know which action is played by Player 2). The observation mapping for the state space for Player 1 is as follows: the set of observations is $\{0, 1, \ldots, n\}$ and we have $\mathsf{obs}_S(s_{\mathsf{init}}) = \mathsf{obs}_S(\mathsf{dead}) = 0$ and $\mathsf{obs}_S(s_{i,j}) = k$ if the j-th variable of C_i is either x_k or its negation $\overline{x}_k$.

A pictorial description is shown in Fig 1. The intuition for the above construction is as follows: Player 2 chooses a clause from the initial state s_{init}, and an observation-based memoryless strategy for Player 1 corresponds to a non-conflicting assignment to the variables. We consider G_Φ with reward functions r, r_1, r_2 as follows: r_2 assigns reward 1 to all transitions; r and r_1 assigns reward 1 to all transitions other than the self-loop at state dead, which is assigned reward 0. We ask the decision questions with $\nu = 1$. Observe that the answer to the decision problems for both mean-payoff and ratio objectives is "Yes" iff the state dead can be avoided by Player 1 (because if dead is reached, then the game stays in dead forever violating both the mean-payoff as well as the raio ojective). We now present the two directions of the proof.

Satisfiable implies dead *is not reached.* We now show that if Φ is satisfiable, then Player 1 has an observation-based memoryless strategy π^* to ensure that dead is never reached. Consider a satisfiable assignment A for Φ, and then the strategy π^* for Player 1 is as follows: given an observation k, if A assigns true to variable x_k, then the strategy π^* chooses action true for observation k, otherwise it chooses action false. Since the assignment A satisfies all clauses, it follows that for every $1 \leq i \leq m$, there exists $s_{i,j}$ such that the strategy π^* for Player 1 ensures that the transition to s_{init} is chosen. Hence the state dead is never reached, and both the mean-payoff and ratio objectives are satisfied.

If dead *is not reached, then* Φ *is satisfiable.* Consider an observation-based memoryless strategy π^* for Player 1 that ensures that dead is never reached. From the strategy π^* we obtain an assignment A as follows: if for observation k, the strategy π^* chooses true, then the assignment A chooses true for variable x_k, otherwise chooses false. Since π^* ensures that dead is not reached, it means for every $1 \leq i \leq m$, that there exists $s_{i,j}$ such that the transition to s_{init} is choosen (which ensures that C_i is satisfied by A). Thus since π^* ensures dead is not reached, the assignment A is a satisfying assignment for Φ.

Thus it follows that the answers to the decision problems are Yes iff Φ is satisfiable, and this establishes the NP-hardness result. ∎

Remark 1: Note that our reduction used only weight values 0 and 1. This implies that NP-hardness holds also for the case where weight values are bounded by a constant.

The NP *upper bounds.* We now present the NP upper bounds for the problem. In both cases, the polynomial witness for the decision problem is a memoryless strategy (i.e., if the answer to the decision problem is Yes, then there is a witness memoryless strategy π for Player 1, and the NP algorithm guesses the witness strategy π). Once the memoryless strategy is guessed and fixed, we need to show that we have a polynomial time verification procedure. The polynomial time verification procedures are as follows:

1) *(Mean-payoff objectives).* Once the memoryless strategy for Player 1 is fixed, the game problem reduces to a problem when there is only Player 2 (this is interpreted as a path problem in directed graphs). The polynomial time algorithm to solve directed graphs with mean-payoff objectives is obtained from the well-known minimum mean-cycle algorithm of [11]. Thus we have the polynomial time verification procedure.
2) *(Ratio objectives).* Again once the memoryless strategy for Player 1 is fixed, the game problem reduces to a problem on directed graphs. It follows from the results of [12] that in directed graphs with ratio objectives memoryless optimal strategies exist (i.e., once the memoryless strategy for Player 1 is fixed, if there is a counter strategy for Player 2, then there is a memoryless one)[2]. Given this fact, we present a simple reduction of the decision problem of ratio objectives to mean-payoff objectives as follows: given reward function r_1 and r_2 and the rational threshold $\nu = \frac{p}{q}$, where $p, q \in \mathbb{N}$; we consider the reward function $r = p \cdot r_1 - q \cdot r_2$, i.e., for every transition $(s\sigma_2\sigma_1 s')$ we have $r(s, \sigma_1, \sigma_2, s') = p \cdot r_1(s, \sigma_1, \sigma_2, s') - q \cdot r_2(s, \sigma_1, \sigma_2, s')$. We consider the reward function r with mean-payoff objective and threshold 0. The mean-payoff objective with reward r and threshold 0 is satisfied iff the ratio objective with rewards r_1 and r_2 is satisfied with threshold ν. Thus from the minimum mean-cycle algorithm of [11], we also obtain a polynomial time verification algorithm for ratio objectives.

We summarize the result in the following theorem.

Theorem 2 (The complexity of the decision problems): The decision problems for partial observation games with mean-payoff objectives and ratio objectives, i.e, whether $\sup_{\pi \in \Pi_{\mathsf{G}}^M} \inf_{\sigma \in \Sigma_{\mathsf{G}}} \mathsf{MPinf}(r)(\rho(s^0, \pi, \sigma)) \geq \nu$ (resp. $\sup_{\pi \in \Pi_{\mathsf{G}}^M} \inf_{\sigma \in \Sigma_{\mathsf{G}}} \mathsf{Ratio}(r_1, r_2)(\rho(s^0, \pi, \sigma)) \geq \nu$), are NP-complete.

[2]It is straightforward to verify that ratio objectives satisfy the criteria of [12] for memoryless optimal strategies.

IV. Real-Time Scheduling of Tasks with Firm Deadlines

Consider a finite set of tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, $n < \infty$, which are all executed on a single processor. Every task τ_i releases countably many task instances (called *jobs*) $J_{i,j}$, $j \geq 1$, over time. All jobs, of all tasks, are independent of each other and can be preempted and resumed during execution without any overhead. We assume a discrete notion of real-time $t = k\varepsilon$, $k \geq 0$, where $\varepsilon > 0$ is both the unit time and the smallest unit of preemption (called a *slot*). Since both task releases and scheduling activities occur at slot boundaries only, all timing values are specified as integers.

A. Basic definitions

Every task τ_i, $1 \leq i \leq n$, is characterized by its non-zero *worst-case execution time* $C_i \in \mathbb{N}$ (slots), its non-zero *relative deadline* $D_i \in \mathbb{N}$ (slots) and its non-zero *utility value* $V_i \in \mathbb{N}$.[3] We will thus sometimes write $\tau_i = (C_i, D_i, V_i)$ to completely specify task τ_i. Every job $J_{i,j}$ of task τ_i needs the processor for C_i (not necessarily consecutive) slots exclusively to execute to completion. All tasks have firm deadlines: Only a job $J_{i,j}$ that completes within D_i slots, as measured from its release time, provides utility to the system. A job that misses its deadline does not do harm but provides zero utility. The real-time scheduling problem considered in this paper is to maximize the *cumulated utility* $V \in \mathbb{N}$ (abbreviated utility), which is the sum of V_i times the number of jobs $J_{i,j}$ that can be completed by their deadlines.

Since there may be slots where no job can be scheduled since no task has been released, we add a special idle task $\tau_0 = (1, 1, V_0)$ with some non-zero $V_0 \in \mathbb{N}$. We assume that a job $J_{0,\ell}$ of τ_0 is released internally at the beginning of every slot $\ell \geq 0$, which may only be executed, however, if no other unfinished job with absolute deadline $> \ell$ has been released earlier. This way, the idle task contributes to the cumulated utility whenever there is no "real" work to be done. This way, the adversary cannot minimize the cumulated utility by not releasing any task.[4]

More formally, let $\Sigma = 2^{\mathcal{T}}$ and consider an infinite *task sequence* $\sigma = (\sigma)^{\ell \geq 0} \in \Sigma^\infty$. For every $\ell \geq 0$, σ^ℓ specifies the subset of tasks a (single) new job of which is released at the beginning of slot ℓ. For $\tau_i \in \sigma^\ell \cup \{\tau_0\}$, job $J_{i,j}$ with $j = |\{k | \tau_i \in \sigma^k \cup \{\tau_0\} \text{ and } 0 \leq k \leq \ell\}|$ is released: $r_{i,j} = \ell$ denotes the *release time* of $J_{i,j}$, which is the earliest time when $J_{i,j}$ can be scheduled and executed, and $d_{i,j} = r_{i,j} + D_i$ denotes its absolute *deadline*.

Given a task sequence σ, the *schedule* $\pi = (\pi)^{\ell \geq 0} \in (\mathcal{T} \cup \{\tau_0\})^\infty$ computed by a real-time scheduling algorithm in the presence of σ is a function that assigns at most one job for execution to every slot $\ell \geq 0$: π^ℓ is either $\emptyset$ (meaning no job is executed) or else some job $J_{i,j}$, which must satisfy the following constraints:

[3]Rational utility values $V_1, \ldots, V_n$ can be mapped to integers by proper scaling.

[4]Alternatively, we could omit the idle task and force the adversary to create infinitely many jobs by adding an appropriate Büchi objective, see Sect. VI.

(i) $r_{i,j} \leq \ell$ (execute only released jobs),
(ii) $\ell < d_{i,j}$ (execute only jobs that could still complete by their deadline),
(iii) $|\{k|\pi^k = J_{i,j} \text{ and } 0 \leq k \leq \ell\}| < C_i$ (execute only jobs that have not been completed),
(iv) $J_{i,j} = J_{0,\ell}$ only if there is no unfinished job $J_{i',j'}$ with $i' > 0$ and $d_{i',j'} > \ell$ (execute idle jobs only if there is no work to be done).[5]

Give some π, let $\pi(k+1)$ denote the prefix of length $k+1$ of π and $\gamma_i(\pi, k+1)$ the number of jobs of task τ_i that are completed by their deadlines in $\pi(k+1)$. The cumulated utility $V(\pi, k+1)$, also called utility for brevity, achieved in $\pi(k+1)$ is defined as $V(\pi, k+1) = \sum_{i=0}^{n} \gamma_i(\pi, k+1)V_i$. Since typically $\lim_{k\to\infty} V(\pi, k+1) = \infty$, however, the utility is not a suitable performance measure for infinite task sequences. We will hence typically focus on the limiting average of the utility (called average utility) $\overline{V}(\pi) = \liminf_{k\to\infty} \frac{1}{k+1} V(\pi, k+1)$ instead. Note that $\overline{V}(\pi)$ is meaningful for characterizing the worst-case cumulated utility of finite task sequences as well, since one can always form an infinite task sequence from a finite one with the same average utility by infinite repetition.

B. Algorithms and Labeled Transition Systems (LTS)

Our main target are deterministic *on-line* scheduling algorithms $\mathcal{A}$, which, at time ℓ, do not know any of the σ^k for $k > \ell$. By contrast, an *off-line* algorithm knows the entire task sequence σ beforehand. Note that, due to our firm deadline assumption, we can restrict our attention to finite-history algorithms here: Let $D_{\max} = \max_{1\leq i\leq n} D_i$ (analogous definitions hold for $C_{\max}$ and $V_{\max}$). Obviously, any job $J_{i,j}$ released by $\ell - D_{\max}$ can be discarded by an algorithm at time ℓ, as scheduling this job would violate (ii). Consequently, algorithms do not need to maintain the full history.

It is easy to express the above setting for a particular on-line algorithm $\mathcal{A}$ as a deterministic labeled transition system $(S, \Sigma_2, \Delta \subseteq S \times \Sigma_2 \times S)$ with reward function r: We choose $\Sigma_2 = \Sigma$ as the set of possible new task releases per slot. Every finite-history algorithm $\mathcal{A}$ can be encoded as a deterministic finite-state automaton with state space S and transition function Δ, which takes the current state and the new jobs released at the beginning of the current slot to compute the new state; the latter also encodes the job $J_{i,j}$ to be executed in the current slot. The reward function $r(\delta)$ for $\delta \in \Delta$ is V_i if $J_{i,j}$ will be completed by the end of the current slot, or 0 otherwise. Given a run $\pi = (\delta)^{\ell\geq 0}$ of the LTS, the average utility is $\overline{V}(\pi) = \liminf_{k\to\infty} \frac{1}{k+1} \sum_{\ell=0}^{k} r(\delta^\ell)$.

Example: LTS for TD1. We illustrate this construction by means of the simple version of the algorithm TD1 introduced in [2, Fig. 1], which is optimal for zero-laxity tasks (i.e., $D_i = C_i$) with uniform value density, i.e., utility values $V_i = C_i$. It maintains three variables

- $v \in \{0, V_1, \ldots, V_n\}$, the utility value of the currently executed job,
- $\Delta \in \{0, \ldots, 4V_{\max}\}$, the time interval between the start of the current busy period and the deadline of the most recently released job (the range follows from the proof of [2, Lem. 5], which reveals $v \geq \Delta/4$),
- $k \in \{0, \ldots, V_{\max} - 1\}$, the number of slots remaining for executing the current job to completion,

all initialized to 0 when the system is/becomes (pre-)idle. Whenever a new job of task $(C_i = V_i, D_i = V_i, V_i)$ is released, TD1 updates Δ to $\Delta := \max\{\Delta, \Delta - k + V_i\}$ and then checks whether $v < \Delta/4$. If so, it discards the current job [if any] and schedules the new job by setting $v := V_i$ and $k = V_i - 1$; otherwise, the current job is not changed and k is just decremented once per slot (if $k > 0$).

If k has reached 0, the current job $J_{i,j}$ has been completed. However, TD1 is not allowed to just switch to the idle state (where the idle task τ_0 adds 1 to the cumulated utility in every slot) in our setting: It may be the case that some jobs have been discarded during $J_{i,j}$'s execution, which have a deadline past $J_{i,j}$'s finishing time. In this case, property (iv) above does not allow the idle task to contribute to the cumulated utility until the deadline of the last discarded job has expired. Hence, our TD1 switches to a *pre-idle* state first, which is the same as the idle state except that the idle task does not add 1 to the cumulated utility. The original TD1 must hence slightly be extended to accommodate this: We add another variable d, $0 \leq d \leq V_{\max} - 1$, initially 0, which is set to $V_i - 1$ upon the arrival of a new job of task τ_i (i.e., when TD1 updates Δ). Like k, d is decremented by 1 in every slot (if $d > 0$); our modified TD1 switches from pre-idle to idle when d reaches 0.

If we denote the states in our state set S as $S(v, \Delta, k, d)$, the transition relation of the corresponding LTS includes e.g. the following states:

- $S(0, 0, 0, 0)$, the idle state,
- $S(0, 0, 0, d)$ with $d > 0$, the pre-idle state with d remaining slots until making the transition to the idle state,
- $S(v, \Delta, 1, d)$ with $v > 0$ and $\Delta > 0$, a state where the current job only needs one additional slot for being completed.

Recalling the ranges of the involved variables, the size of the state space S is $|S| = (1 + 4nV_{\max}^2)V_{\max}$. For $C_i = D_i = V_i = i$ and hence $V_{\max} = n$, which is the task set that allows comparison of our results with [2], the above version of TD1 hence needs $4n^3 + n$ states.[6]

Every transition $\delta \in S \times \Sigma \times S$ of our LTS corresponds to the execution of a single slot, and depends on the jobs arriving at the beginning of this slot. In general, transitions must be

[5]Note carefully that this definition does not allow the idle task to create utility as long as there is a regular task in the system with a deadline that has not yet expired - even if this task will eventually not be able to make its deadline because of an excessive remaining execution time. We cannot allow the idle task to already contribute utility when it becomes clear that the remaining tasks will not make their deadlines, though: An optimal on-line algorithm would then just discard any arriving job immediately, and still create 100% of the achievable cumulated utility by means of the idle task.

[6]There is certainly room for improvement here, by using a more efficient encoding of the state.

labeled with the *sets* of newly arriving jobs σ^ℓ. In case of TD1, however, it is easy to see (cf. the update-rule for Δ) that it suffices to label transitions just with the task τ_i with maximum utility V_i in σ^ℓ. We list a few example transitions of TD1 (we write $S \xrightarrow{\tau_1} S'$ for $(S, \tau_1, S') \in \Delta$ for clarity):

- For $d \geq 0$:
  $S(0,0,0,d) \xrightarrow{\tau_i} S(V_i, V_i, V_i - 1, \max\{d, V_i - 1\})$: τ_i initiates new busy period.
- For $k > 1$, $d > 1$, $V_i \geq (\Delta - k + V_j)/4$:
  $S(V_i, \Delta, k, d) \xrightarrow{\tau_j} S(V_i, \Delta - k + V_j, k - 1, d - 1)$: TD1 retains current task but does not complete it.
- For $d > 0$, $V_i \geq (\Delta - k + V_j)/4$:
  $S(V_i, \Delta, 1, d) \xrightarrow{\tau_j} S(0,0,0,d-1)$: TD1 retains current task and does complete it (adds V_i to cumulated utility).
- $S(0,0,0,1) \rightarrow S(0,0,0,0)$: Move from pre-idle to idle (do not add 1 to cumulated utility).
- $S(0,0,0,0) \rightarrow S(0,0,0,0)$: Execute idle task (add 1 to cumulated utility).
- For $k > 1$, $d > 1$, $V_i < (\Delta - k + V_j)/4$:
  $S(V_i, \Delta, k, d) \xrightarrow{\tau_j} S(V_j, \Delta - k + V_j, V_j - 1, V_j - 1)$: TD1 discards current task and starts new one.

The non-deterministic LTS. Obviously, except for computing the average utility, tailoring our game model to a particular algorithm would give away most of its power. For competitive analysis, we will thus utilize a *non-deterministic* LTS, which can simulate all possible on-line algorithms in a memoryless way. In case of our real-time scheduling problem, such an LTS for a given task set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ can be easily constructed, since property (ii) allows to discard all jobs that have been released $\geq D_{\max}$ slots ago. Its state space hence only has to encode, for the at most $D_{\max}$ jobs of each $\tau_1, \ldots, \tau_n$ arriving during the $D_{\max}$ most recent slots, every job's current state.

In our non-deterministic LTS, ihe state of any job $J_{i,j}$ consists of

- $k \in \{0 \ldots, C_i\}$, the number of slots of $J_{i,j}$ that still remain to be executed,
- $d \in \{0, \ldots, D_i - 1\}$, the number of slots of $J_{i,j}$ remaining until $d_{i,j}$ expires (the job's laxity).

Note that $(0, d)$ denotes a state where $J_{i,j}$ has been completed, with d slots remaining until its deadline expires. Similarly, $(k, 0)$ with $k > 0$ represents a job that did not make its deadline.

The entire state space S of our LTS consists of an array $J[i, \ell]$ of D_{max} columns and n rows. Row i, $1 \leq i \leq n$, represents task τ_i, and column ℓ, $1 \leq \ell \leq D_{\max}$, represent a slot. The columns are organized as a circular buffer for the jobs arriving in the most recent $D_{\max}$ slots: S also contains an index variable $\ell \in \{1, \ldots, D_{\max}\}$, initially 1, which is incremented $\mod D_{\max} + 1$ at the end of every slot. If $\sigma \subseteq 2^{\mathcal{T}}$ denotes the tasks a new job of which is released at the beginning of the current slot, every state transition atomically performs the following actions:

1) $\forall \tau_i \in \sigma : J[i, \ell] := (C_i, D_i)$ (record newly arriving jobs, if any).
2) Non-deterministically choose some index $[i', \ell']$ where $J[i', \ell'] = (k', d')$ with both $k' > 0$ and $d' > 0$, and set $J[i', \ell'] := (k' - 1, d')$ (schedule a slot of $J[i', \ell']$ with non-zero remaining processing time for execution).
3) For all $i, 1 \leq i \leq n, 1 \leq m \leq D_{\max}$ where $J[i, m] = (., d)$ with $d > 0$, set $J[i, m] := (., d - 1)$ (decrease laxity of all jobs with non-zero laxity).

The transition relation of our LTS contains *all* transitions adhering to the above, for all σ. A transition δ where $J[i', \ell']$ is set to $(0, d)$ represents the completion of a job of $\tau_{i'}$; its reward is $r(\delta) = V_{i'}$. All other transitions (except idle transitions, see below) have reward 0.

As in the LTS for TD1, we must distinguish between the real idle states of the system (one for every ℓ), where all entries of J are $(0, 0)$, and pre-idle states, where at least one entry of J is of the form (k, d) with $d > 0$. Transitions δ from an idle state to an idle state have reward $r(\delta) = 1$, representing the utility value of our idle task τ_0. Transitions from a pre-idle state to a pre-idle or idle state have reward 0.

Figure 2 shows two (non-deterministic) transitions $\delta' = (s, \sigma, s')$ and $\delta'' = (s, \sigma, s'')$ from state s into successor states s' and s'', respectively, including the three internal actions performed during these transitions. We assume a task set consisting of three tasks here:

$$\begin{array}{ll} \tau_1: & C_1 = 2, D_1 = 2, V_1 = 2, \\ \tau_2: & C_2 = 1, D_2 = 1, V_2 = 1, \text{and} \\ \tau_3: & C_3 = 2, D_3 = 3, V_3 = 2. \end{array}$$

State s is such that the first job of τ_1 and τ_3 have already been released in the slot before (where $J_{1,1}$ has been chosen for execution), such that $\ell = 2$. In the slot corresponding to δ' resp. δ'', we assume that two additional jobs arrive: the second instance of task τ_1 and the first instance of task τ_2, i.e., $\sigma = \{\tau_1, \tau_2\}$. In the second action, the non-deterministic choice happens. There are actually four possibilities here, as there are four jobs ready for being scheduled in the system. The label on each arrow gives the entry $[i', \ell']$ of the job being chosen. In the third action, the remaining laxity is decreased for all jobs in the system, which concludes the state transition. Transition δ'', leading to s'', chooses entry $[2, 2]$ in the second action and thus gains reward $r(\delta'') = V_2 = 1$, as the job is completed. On the other hand, transition δ', leading to s', chooses job $[1, 2]$ and thus gains reward $r(\delta') = 0$.

Obviously, this non-deterministic finite-state LTS can simulate any possible real-time scheduling algorithm with a memoryless strategy, as all required history information is encoded in the state. Taking into account that every of the $nD_{\max}$ job entries in J can take on at most $(C_{max}+1)D_{\max}$ different states, and the fact ℓ can take on $D_{\max}$ different values, the overall size of the state space is $|S| = D_{\max}((C_{max}+1)D_{\max})^{nD_{\max}}$. We hence arrive at the following result.

Theorem 3 (LTS reduction of real-time scheduling problems): There is a reduction of every real-time scheduling problem with firm deadlines to a non-deterministic LTS with $|S| = D_{\max}((C_{max} + 1)D_{\max})^{nD_{\max}}$ states, which can

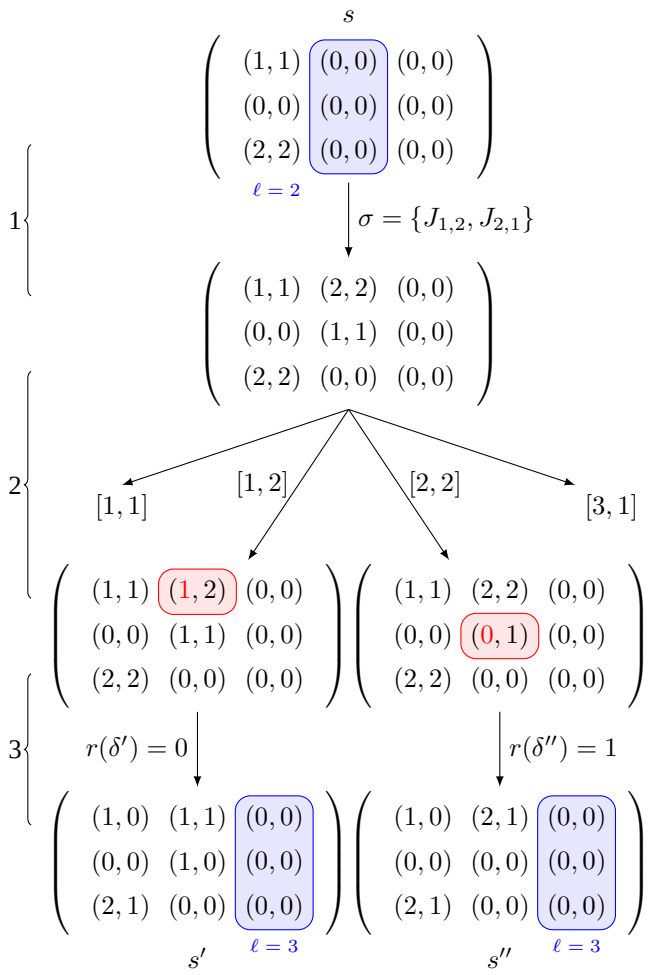

Fig. 2. Example of two non-deterministic LTS transitions $\delta' = (s, \sigma, s')$ and $\delta'' = (s, \sigma, s'')$, from state s into successor states s' and s'', respectively, including the three internal actions performed during these transitions, for a task set consisting of three tasks. δ'' chooses entry $[2, 2]$ and thus gains reward $r(\delta'') = V_2 = 1$, δ' chooses job $[1, 2]$ and thus gains reward $r(\delta') = 0$.

simulate any real-time scheduling algorithm for this problem adhering to properties (i)–(iv) with a memoryless strategy.

V. Games for Real-time Scheduling Analysis

A. Worst-case average utility

Before turning our attention to the competitive analysis of our real-time scheduling algorithms, i.e., the competitive ratio [2], we first establish general results on worst-case average utilities. They follow easily from the powerful framework of perfect-information graph games (Theorem 1).

We have already shown that our scheduling problem can be reduced to a non-deterministic finite-state LTS $(S, \Sigma_2, \Delta \subseteq S \times \Sigma_2 \times S)$ along with a reward function r, such that the mean-payoff function $\mathsf{MPinf}(r)$ gives the average utility. We simply interpret the transition system as a game, where the choices of transitions δ^G correspond to the actions for Player 1. Thus we have a perfect-information game, and every memoryless strategy correspond to a scheduling algorithm and vice-versa (i.e., every required scheduling algorithm is a memoryless strategy of the game). The worst-case utility of a given on-line algorithm, corresponding to a memoryless strategy $\pi \in \Pi_{\mathsf{G}}^M$, is

$$\inf_{\sigma \in \Sigma_{\mathsf{G}}} \mathsf{MPinf}(r)(\rho(s^0, \pi, \sigma)). \tag{1}$$

Note that it can be computed from the LTS of the particular on-line algorithm. The worst-case utility of the optimal on-line algorithm is given by

$$\sup_{\pi \in \Pi_{\mathsf{G}}^M} \inf_{\sigma \in \Sigma_{\mathsf{G}}} \mathsf{MPinf}(r)(\rho(s^0, \pi, \sigma)); \tag{2}$$

its computation requires the non-deterministic LTS of the real-time scheduling problem. Using the results of Theorem 1 we obtain the following result.

Theorem 4: The following assertions hold for the class of real-time scheduling problems defined in Sect. IV:

1) The worst-case average utility for a given on-line algorithm can be in time $O(|S| \cdot m \cdot \log V_{\max})$, where $|S|$ resp. m is the number of states resp. transitions of the deterministic LTS of the algorithm and $V_{\max}$ is maximum utility value of any task.
2) Whether there exists an online algorithm with worst-case average utility at least ν can be decided in NP∩ coNP in general; and if $V_{\max}$ is bounded by the size of the non-deterministic LTS, then the decision problem can be solved in polynomial time.
3) An online algorithm with optimal worst-case average utility can be constructed in time $O(|S| \cdot m \cdot V_{\max})$, where $|S|$ resp. m is the number of states resp. transitions of the non-deterministic LTS.

Theorem 1 also allows us to derive an interesting result on the *worst-case utility* ratio.

Worst-case utility ratio. The worst-case utility ratio is the worst-case limiting average utility of the best online algorithm over the worst-case limiting average utility achievable by the best offline algorithm.

Worst-case utility means the smallest cumulated utility that can be guaranteed by the best (and hence by any) algorithm in the presence of the worst-case task sequence. Note that defining the worst-case *utility* ratio in terms of a ratio of limiting *average utilities* is possible here, as the leading factor of $1/k$ in the latter cancels out in the ratio. In contrast to the competitive ratio, the worst-case utility ratio relates the average utilities of the on-line and off-line algorithm for possibly *different* task sequences. It is not difficult to prove, however, that the worst-case utility ratio is always an upper bound on the competitive factor of an optimal on-line algorithm.

Recalling (2), the worst-case utility of an offline algorithm is given by

$$\inf_{\sigma \in \Sigma_{\mathsf{G}}} \sup_{\pi \in \Pi_{\mathsf{G}}^M} \mathsf{MPinf}(r)(\rho(s^0, \pi, \sigma)). \tag{3}$$

Observe that the online algorithm makes the choice of strategy without knowing in advance the strategy of the adversary, whereas the offline algorithm makes the choice after the input sequence has been choosen by the opponent. Theorem 1 also provide the following result for the ratio of (2) and (3).

Theorem 5: The worst-case utility ratio for the class of real-time scheduling problems defined in Sect. IV is always 1.

Finally, according to Theorem 4, it is also possible to compute the worst-case utility ratio for a given on-line algorithm, by computing the ratio of (1) and (3).

B. Competitive analysis

We now show how the competitive ratio analysis can be reduced to the partial-observation game problem with ratio objectives. It follows from the results of Theorem 2 that the decision problem for competitive-ratio can be solved in NP.

Labeled transition system to partial-observation game. Given our non-deterministic LTS $(S, \Sigma_2, \Delta \subseteq S \times \Sigma_2 \times S)$ with reward function r, we first construct the equivalent perfect-information game $(S, \Sigma_1, \Sigma_2, \delta)$. Σ_1 represent the choice of the transition function as actions and $\delta : S \times \Sigma_1 \times \Sigma_2 \to S$ gives the transition. We now construct a partial-observation game G as follows: $\mathsf{G} = (S^G = S \times S, \Sigma_1^G = \Sigma_1, \Sigma_2^G = \Sigma_2 \times \Sigma_1, \delta^G, \mathcal{O}_S, \mathcal{O}_\Sigma)$. Intuitively, we construct the product game with two components, where Player 1 only observes the first component and makes choice of the transitions there; and Player 2 is in charge of choosing the input and also the transitions of the second component. However, due to partial observation, Player 1 does not observe the choice of transitions in the second component. We describe the transition and the observation mapping to capture this: (i) the transition function $\delta^G : S^G \times \Sigma_1^G \times \Sigma_2^G \to S^G$ is as follows: $\delta^G((s_1, s_2), \sigma_1, (\sigma_2, \overline{\sigma}_1)) = (\delta(s_1, \sigma_1, \sigma_2), \delta(s_2, \sigma_2, \overline{\sigma}_1))$; (ii) the observation for states for Player 1 maps every state to the first component, i.e., $\mathsf{obs}_S((s_1, s_2)) = s_1$ and the observation for actions for Player 1 maps every state to the first component as well (i.e., the input from Player 2), i.e., $\mathsf{obs}_\Sigma((\sigma_2, \overline{\sigma}_1)) = \sigma_2$. The two reward functions are as follows: the reward function r_1 gives reward according to r and the transitions of the first component and the reward function r_2 assigns reward according to r and the transitions of the second component. Note that the construction ensures that we compare the utility of an online algorithm and an offline algorithm (chosen by Player 2 using the second component) that operate on the same input sequence. It follows that there is an online algorithm with competitive-ratio at least ν iff $\sup_{\pi \in \Pi_{\mathsf{G}}^M} \inf_{\sigma \in \Sigma_{\mathsf{G}}} \mathsf{Ratio}(r_1, r_2)(\rho(s^0, \pi, \sigma)) \geq \nu$. Note that in the ratio objective we add the value 1 in the denominator and numerator to ensure that a division by zero cannot occur. Since our choice of rewards in conjunction with the idle task ensures that the sum of the rewards in the denominator goes to infinity, adding the constant value 1 does not affect the competitive ratio analysis. By Theorem 2 the decision can be achieved in NP in the size of the LTS.

Theorem 6: For the class of scheduling problems defined in Sect. IV, the competitive analysis can be achieved in NP in the size of the LTS (constructed from the scheduling problem).

VI. Extensions

In the previous section, we presented results of applying partial-observation games and perfect-observation games with mean-payoff and ratio objectives to our real-time scheduling problem. In this section, we show that it is easy to add various constraints on the adversary to the original setting, without unduly increasing the complexity of the involved decision problems. This particularly attractive feature of our approach follows from the very flexible framework of games, which allows to combine mean-payoff and ratio objectives with other classes of Boolean and quantitative objectives.

1) *Safety objectives.* In general, safety objectives ensure that certain states of the game are never reached in any play. This is accomplished by penalizing the transitions to/from a forbidden state by a prohibitively large weight, such that the adversary can never visit it without losing the game. All our complexity results continue to hold, because the underlying (polynomial time) graph algorithms also work with additional safety conditions.
   In our real-time scheduling context, safety objectives can e.g. be used to constrain the maximum load created by the adversary by some threshold u. All that is needed to accomplish this is to keep track of the actual load $u(t)$ within the state of the LTS, and to go to a winning state for Player 1 whenever $u(t) > u$. A more elaborate application of safety objectives can be found in energy-constrained real-time scheduling, where the goal is to maximize the cumulated utility achievable with some fixed energy budget E. In [8], the adversary must be constrained to only generate task sequences that can be feasibly scheduled (without any deadline violation) for $E = \infty$. Since EDF is known to be an optimal scheduling algorithm for the latter problem, it suffices to also incorporate an instance of EDF into our LTS that leads to a winning state for Player 1 in case of a deadline violation.
2) *Büchi objectives.* We can also extend our results with additional Büchi objectives, which ensure that certain states are visited infinitely often in every play. Again, our results continue to hold, as the underlying graph algorithms just need to first consider strongly connected components with Büchi states and then to apply the mean-payoff graph algorithm to ensure that the mean-payoff cycle contains at least one Büchi state.
   In our real-time scheduling context, Büchi objectives can e.g. be used to constrain the adversary to generate only finite periods of overload. This can be achieved by adding the requirement that idle states of the LTS are visited infinitely often. Another application of Büchi objectives is for modeling semi-online algorithms, which assume that the scheduling algorithm knows something about the future behavior. A typical example is that the adversary will eventuall generate an instance of a

particular task (e.g., with some given execution time C_i [8]).

3) *Multi-dimensional objectives.* We can also extend our standard setting with multi-dimensional objectives, where instead of one mean-payoff objective there are multiple ones. Again, all our results also extend to such objectives, since graphs with multiple mean-payoff objectives can also be solved in polynomial time [7], [17].

   This extension could be used in our real-time scheduling context for e.g. constraining the average load—rather than the maximum load—generated by the adversary during some time interval. Multi-dimensional objectives also allow to incorporate additional cost functions. For example, minimizing energy consumption, in addition to maximizing cumulated utility, is an important concern in modern real-time systems [1].

In this paper, lacking space does of course not allow us to address these extensions in detail, not to speak of exhaustively. After all, there is a considerable body of existing work on competitive analysis of real-time scheduling algorithms, which addresses many different additional properties/goals: Energy consumption [1], [8] (including dynamic voltage scaling), imprecise computation tasks (having both a mandatory and an optional part and associated utilities) [4], lower bounds on slack time [3], and fairness [14]. We are convinced, however, that it is not too difficult to incorporate and analyze even sophisticated extensions without much additional efforts.

VII. Conclusions

We showed how to employ the powerful tool of algorithmic game theory for competitive analysis of real-time scheduling algorithms for tasks with firm deadlines. We developed a novel partial-observation game suitable for this purpose, and showed that all involved problems of interest are decideable. Comparison with [2] reveals the following advantages/disadvantages of our approach: On the positive side, as an algorithmic approach, our solution does not involve human ingenuity. Moreover, apart from the fact that it deals with arbitrary task utility values V_i right from the start, i.e., not just uniform utilities $V_i = C_i$, our method can easily accomodate various additional constraints on the adversary. Part of our future work will be devoted to exploring these extensions in detail.

On the negative side, whereas computing the worst-case cumulated utility etc. of a *given* on-line algorithm can usually be done in polynomial time, the practical utility of our approach for automated construction of an optimal on-line algorithm and competitive analysis is severely impaired by the exponential time complexity involved in the LTS reduction (Theorem 3) and the competitive analysis decision problem (Theorem 2). A central part of our future work will hence be devoted to coping with the complexity issue, both by identifying relevant problem sub-classes with polynomial complexity and by trying to find heuristics that allow to efficiently deal with (most) instances of the general problem.

Another limitation of our approach, which is irrelevant in practice, though, follows from the required finiteness of the state space of the LTS (without which all problems of interest become undecideable). In particular, we cannot allow the number of tasks n to become infinite. Consequently, we could not prove $1/4$ to be the competitive factor for uniform utilities $V_i = C_i$, as this effectively requires infinitely many tasks [2].[7]

Overall, our findings reveal that automated competitive analysis using algorithmic game theory is principally feasible, and possibly opens up an interesting direction of further real-time systems research.

References

[1] Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejía-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. Comput.*, 53(5):584–600, May 2004.

[2] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Syst.*, 4:125–144, May 1992.

[3] Sanjoy K. Baruah and Jayant R. Haritsa. Scheduling for overload in real-time systems. *IEEE Trans. Comput.*, 46(9):1034–1039, September 1997.

[4] Sanjoy K. Baruah and Mary Ellen Hickey. Competitive on-line scheduling of imprecise computations. *IEEE Transactions on Computers*, 47:1027–1032, 1998.

[5] Allan Borodin and Ran El-Yaniv. *Online COmputation and Competitive Analysis*. Cambridge University Press, 1998.

[6] Lubos Brim, Jakub Chaloupka, Laurent Doyen, Raffaella Gentilini, and Jean-François Raskin. Faster algorithms for mean-payoff games. *Formal Methods in System Design*, 38(2):97–118, 2011.

[7] Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Generalized mean-payoff and energy games. In *FSTTCS*, pages 505–516, 2010.

[8] Vinay Devadas, Fei Li, and Hakan Aydin. Competitive analysis of online real-time scheduling algorithms under hard energy constraint. *Real-Time Syst.*, 46(1):88–120, September 2010.

[9] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *Int. Journal of Game Theory*, 8(2):109–113, 1979.

[10] V. A. Gurvich, A. V. Karzanov, and L. G. Khachivan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *USSR Comput. Math. Math. Phys.*, 28(5):85–91, April 1990.

[11] R.M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[12] E. Kopczynski. Half-positional determinacy of infinite games. In *ICALP (2)*, pages 336–347, 2006.

[13] D. A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

[14] Michael A. Palis. Competitive algorithms for fine-grain real-time scheduling. *Real-Time Systems Symposium, IEEE International*, 0:129–138, 2004.

[15] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1993.

[16] L.S. Shapley. Stochastic games. *Proc. Nat. Acad. Sci. USA*, 39:1095–1100, 1953.

[17] Yaron Velner. The complexity of mean-payoff automaton expression. In *ICALP*, 2012.

[18] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.

[7] What we can prove, however, is a competitive factor of $1/4 + \epsilon(n)$, for some $\epsilon(n) > 0$, by applying our approach to the task set $\mathcal{T} = \{\tau_i = (i, i, i) | \ 1 \leq i \leq n\}$.