Real-Time Syst (2018) 54:166–207
https://doi.org/10.1007/s11241-017-9293-4

CrossMark

Automated competitive analysis of real-time scheduling with graph games

Krishnendu Chatterjee[1] · Andreas Pavlogiannis[1] · Alexander Kößler[2] · Ulrich Schmid[2]

Published online: 1 November 2017

Abstract This paper is devoted to automatic competitive analysis of real-time scheduling algorithms for firm-deadline tasksets, where only completed tasks contribute some utility to the system. Given such a taskset $\mathcal{T}$, the competitive ratio of an on-line scheduling algorithm $\mathcal{A}$ for $\mathcal{T}$ is the worst-case utility ratio of $\mathcal{A}$ over the utility achieved by a clairvoyant algorithm. We leverage the theory of quantitative graph games to address the *competitive analysis* and *competitive synthesis* problems. For the competitive analysis case, given any taskset $\mathcal{T}$ and any finite-memory on-line scheduling algorithm $\mathcal{A}$, we show that the competitive ratio of $\mathcal{A}$ in $\mathcal{T}$ can be computed in polynomial time in the size of the state space of $\mathcal{A}$. Our approach is flexible as it also provides ways to model meaningful constraints on the released task sequences that determine the competitive ratio. We provide an experimental study of many well-known on-line scheduling algorithms, which demonstrates the feasibility of our competitive analysis approach that effectively replaces human ingenuity (required

Preliminary versions of this paper have appeared in Chatterjee et al. (2013, 2014).

✉ Andreas Pavlogiannis
pavlogiannis@ist.ac.at

Krishnendu Chatterjee
krish.chat@ist.ac.at

Alexander Kößler
koe@ecs.tuwien.ac.at

Ulrich Schmid
s@ecs.tuwien.ac.at

1 IST Austria (Institute of Science and Technology Austria), Am Campus 1, 3400 Klosterneuburg, Austria

2 Embedded Computing Systems Group, Vienna University of Technology, Treitlstrasse 3, 1040 Vienna, Austria

for finding worst-case scenarios) by computing power. For the competitive synthesis case, we are just given a taskset $\mathcal{T}$, and the goal is to automatically synthesize an optimal on-line scheduling algorithm $\mathcal{A}$, i.e., one that guarantees the largest competitive ratio possible for $\mathcal{T}$. We show how the competitive synthesis problem can be reduced to a two-player graph game with partial information, and establish that the computational complexity of solving this game is NP-complete. The competitive synthesis problem is hence in NP in the size of the state space of the non-deterministic labeled transition system encoding the taskset. Overall, the proposed framework assists in the selection of suitable scheduling algorithms for a *given* taskset, which is in fact the most common situation in real-time systems design.

Keywords Real-time scheduling · Firm-deadline tasks · Competitive analysis · Quantitative graph games

1 Introduction

We study the well-known problem of scheduling a sequence of dynamically arriving real-time task instances with firm deadlines on a single processor, by using automatic solution techniques based on graphs and games. In firm-deadline scheduling, a task instance (a *job*) that is completed by its deadline contributes a positive utility value to the system; a job that does not meet its deadline does not harm, but does not add any utility. The goal of the scheduling algorithm is to maximize the cumulated utility. Firm-deadline tasks arise in various application domains, e.g., machine scheduling (Gupta and Palis 2001), multimedia and video streaming (Abeni and Buttazzo 1998), QoS management in bounded-delay data network switches (Englert and Westermann 2007) and even networks-on-chip (Lu and Jantsch 2007), and other systems that may suffer from overload (Koren and Shasha 1995).

Competitive analysis (Borodin and El-Yaniv 1998) has been the primary tool for studying the performance of such scheduling algorithms (Baruah et al. 1992). It allows to compare the performance of an *on-line* algorithm $\mathcal{A}$, which processes a sequence of inputs without knowing the future, with what can be achieved by an optimal *off-line* algorithm $\mathcal{C}$ that does know the future (a *clairvoyant* algorithm): the *competitive factor* gives the worst-case performance ratio of $\mathcal{A}$ vs. $\mathcal{C}$ over all possible scenarios.

In a seminal paper, Baruah et al. (1992) proved that no on-line scheduling algorithm for single processors can achieve a competitive factor better than $1/4$ over a clairvoyant algorithm in *all* possible job sequences of *all* possible tasksets. The proof is based on constructing a specific job sequence, which takes into account the on-line algorithm's actions and thereby forces any such algorithm to deliver a sub-optimal cumulated utility. For the special case of zero-laxity tasksets of uniform value-density (where utilities equal execution times), they also provided the on-line algorithm TD1 with competitive factor $1/4$, concluding that $1/4$ is a tight bound for this family of tasksets. In Baruah et al. (1992), the $1/4$ upper bound was also generalized, by showing that there exist tasksets with *importance ratio* k, defined as the ratio of the maximum over the minimum value-density in the taskset, in which no on-line scheduler can have competitive factor larger than $\frac{1}{(1+\sqrt{k})^2}$. In subsequent work (Koren and Shasha

1995), the on-line scheduler D^{over} was introduced, which provides the performance guarantee of $\frac{1}{(1+\sqrt{k})^2}$ in any taskset with importance ratio k, showing that this bound is also tight.

1.1 Problems considered in this paper

Since the taskset arising in a particular application is usually known, the present work focuses on the competitive analysis problem for *given* tasksets: rather than from *all* possible tasksets as in Baruah et al. (1992), the job sequences used for determining the *competitive ratio* are chosen from a taskset given as an input. Note that this is in fact the most common situation faced by real-time system *designers*, which would clearly welcome automatic techniques in the first place. We hence study the two relevant problems for the *automated* competitive analysis for given tasksets:

(1) The *competitive analysis* question asks to compute the competitive ratio of a given on-line algorithm.
(2) The *competitive synthesis* question asks to construct an on-line algorithm with optimal competitive ratio.

Both question are relevant in online-scheduling settings where the taskset is known in advance. The competitive analysis problem can determine the performance of existing schedulers, and help with choosing the one that is best in the given setting. The competitive synthesis problem can even provide a scheduler that is optimal by construction in the given setting.

1.2 Detailed contributions

Our contributions on each problem are as follows.

1.2.1 Competitive analysis

Given a taskset $\mathcal{T}$ and an on-line scheduling algorithm $\mathcal{A}$, the competitive analysis question asks to determine the competitive ratio of $\mathcal{A}$ when the arriving jobs are instances of tasks from $\mathcal{T}$. Our respective results are provided in the following sections:

- In Sect. 2, we formally define our real-time scheduling problem.
- In Sect. 3, we provide a formalism for on-line and clairvoyant scheduling algorithms as labeled transitions systems. We also show how automata on infinite words can be used to express natural constraints on the set of released job sequences (such as sporadicity and workload constraints).
- In Sects. 4.1 and 4.2, we define graph objectives on weighted multi-graphs and provide algorithms for solving those objectives.
- In Sect. 4.3, we present a formal reduction of the competitive analysis problem to solving a multi-objective graph problem. Section 4.4 describes both general and implementation-specific optimizations for the above reduction, which considerably reduce the size of the obtained graph and thus make our approach feasible in practice.

– In Sect. 4.5, we present a comparative study of the competitive ratio of several existing firm-deadline real-time scheduling algorithms. Our results show that the competitive ratio of any algorithm varies highly when varying tasksets, which highlights the usefulness of an automated competitive analysis framework: after all, our framework allows to replace human ingenuity (required for finding worst-case scenarios) by computing power, as the application designer can analyze different scheduling algorithms for the specific taskset arising in some application and compare their competitive ratio.

1.2.2 Competitive synthesis

Given a taskset $\mathcal{T}$, the competitive synthesis question asks to construct an on-line scheduling algorithm $\mathcal{A}$ with optimal competitive ratio for $\mathcal{T}$: the competitive ratio of $\mathcal{A}$ for $\mathcal{T}$ is at least as large as the competitive ratio of any other on-line scheduling algorithm for $\mathcal{T}$. Our respective results are presented in Sect. 5:

– In Sect. 5.1, we consider a game model (a partial-observation game with memoryless strategies for Player 1 with mean-payoff and ratio objectives) that is suitable for the competitive synthesis of real-time scheduling algorithms. The mean-payoff (resp. ratio) objective allows to compute the cumulated utility (resp. competitive ratio) of the best on-line algorithm under the worst-case task sequence.
– In Sect. 5.2, we establish that the relevant decision problems for the underlying game are NP-complete in the size of the game graph.
– In Sect. 5.3, we use the game of Sect. 5.1 to tackle two relevant synthesis problems for a given taskset $\mathcal{T}$: first, we show that constructing an on-line scheduling algorithm with optimal worst-case average utility for $\mathcal{T}$ is in NP ∩ coNP in general, and polynomial in the size of the underlying game graph for reasonable choices of task utility values. Second, we show that constructing an on-line scheduling algorithm with optimal competitive ratio for $\mathcal{T}$ is in NP. Note that these complexities are with respect to the size of the constructed algorithm, represented explicitly as a labeled transition system. As a function of the input taskset $\mathcal{T}$ given in binary, all polynomial upper bounds become exponential upper bounds in the worst case.

1.3 Related work

Algorithmic game theory (Nisan et al. 2007) has been applied to classic scheduling problems since decades, primarily in economics and operations research, see e.g. (Koutsoupias 2011) for just one example of some more recent work. In the real-time systems context, *mechanism design* (Porter 2004) is an area where game theory is actually the method of choice: rather than determining the performance of a scheduling algorithm resp. finding an optimal one for some given taskset, i.e., for some given set of rules, which is our goal, the challenge in mechanism design is to define the rules that allow the system to to achieve certain goals, e.g., performance, in the presence of rational agents that strive for maximizing some local benefit.

However, game theory has also been applied to problems that are more closely related to the one studied in this paper. In particular, Sheikh et. al. (2011) considered

the problem of non-preemptively scheduling periodic *hard* real-time tasks (where all jobs must make their deadlines), and used an optimal strategy in the context of non-cooperative games to optimally determine the initial offsets of all tasks in the periodic schedule. Altisen et al. (2002) used games for synthesizing controllers dedicated to meeting all deadlines in systems with shared resources. Bonifaci and Marchetti-Spaccamela (2012) employed graph games for automatic feasibility analysis of sporadic real-time tasks in multiprocessor systems: given a set of sporadic tasks (where consecutive releases of jobs of the same task are separated at least by some sporadicity interval), the algorithms provided in Bonifaci and Marchetti-Spaccamela (2012) allow to decide, in polynomial time, whether some given scheduling algorithm will meet *all* deadlines. A partial-information game variant of their approach also allows to synthesize an optimal scheduling algorithm for a given taskset (albeit not in polynomial time).

A recent work (Lübbecke et al. 2016) studies the synthesis of schedulers where the task is to minimize the weighted sum of completion times of the released tasks. It is shown how the competitive ratio can be approximated in various online schemes, e.g. for parallel, related, and unrelated machines. We note that our online setting differs on the objective function that needs to be obtained (i.e., maximizing utility vs minimizing completion times).

Regarding firm-deadline task scheduling in general, starting out from (Baruah et al. 1992), classic real-time systems research has studied the competitive factor of both simple and extended real-time scheduling algorithms. The competitive analysis of simple algorithms has been extended in various ways later on: energy consumption (Aydin et al. 2004; Devadas et al. 2010) (including dynamic voltage scaling), imprecise computation tasks (having both a mandatory and an optional part and associated utilities) (Baruah and Hickey 1998), lower bounds on slack time (Baruah and Haritsa 1997), and fairness (Palis 2004). Note that dealing with these extensions involved considerable ingenuity and efforts w.r.t. identifying and analyzing appropriate worst-case scenarios, which do not necessarily carry over even to minor variants of the problem. Maximizing cumulated utility while satisfying multiple resource constraints is also the purpose of the Q-RAM (QoS-based Resource Allocation Model) (Rajkumar et al. 1997) approach.

Preliminary versions of this work have appeared in Chatterjee et al. (2014) (competitive analysis) and (Chatterjee et al. 2013) (competitive synthesis). The present paper unifies the two topics in a common framework, which we develop in more detail. Additionally, we have extended our experiments to also incorporate the well-known scheduling algorithms TD1 (Baruah et al. 1992) and least laxity first (Leung 1989). Finally, we have extended our competitive synthesis approach to also cover constrained environments (imposing safety, liveness and limit-average constraints for the generated job sequences).

2 Problem definition

We consider a finite set of tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_N\}$, to be executed on a single processor. We assume a discrete notion of real-time $t = k\varepsilon$, $k \geqslant 1$, 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 positive integers. Every task τ_i releases countably many task instances (called *jobs*) $J_{i,j} := (\tau_i, j) \in \mathcal{T} \times \mathbb{N}^+$ (where $\mathbb{N}^+$ is the set of positive integers) over time (i.e., $J_{i,j}$ denotes that a job of task i is released at time j). All jobs, of all tasks, are independent of each other and can be preempted and resumed during execution without any overhead. Every task τ_i, for $1 \leqslant i \leqslant N$, is characterized by a 3-tuple $\tau_i = (C_i, D_i, V_i)$ consisting of 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* $V_i \in \mathbb{N}^+$ (rational utility values $V_1, \ldots, V_N$ can be mapped to integers by proper scaling). We denote with $D_{\max} = \max_{1 \leqslant i \leqslant N} D_i$ the maximum relative deadline in $\mathcal{T}$. Every job $J_{i,j}$ 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 V_i to the system. A job that misses its deadline does not harm but provides zero utility. The goal of a real-time scheduling algorithm in this model is to maximize the *cumulated utility*, which is the sum of V_i times the number of jobs $J_{i,j}$ that can be completed by their deadlines, in a sequence of job releases generated by the *adversary*.

2.1 Notation on sequences

Let X be a finite set. For an infinite sequence $x = (x^\ell)_{\ell \geqslant 1} = (x^1, x^2, \ldots)$ of elements in X, we denote by x^ℓ the element in the ℓ-th position of x, and denote by $x(\ell) = (x^1, x^2, \ldots, x^\ell)$ the finite prefix of x up to position ℓ. We denote by X^∞ the set of all infinite sequences of elements from X. Given a function $f : X \to \mathbb{Z}$ (where $\mathbb{Z}$ is the set of integers) and a sequence $x \in X^\infty$, we denote with $f(x, k) = \sum_{\ell=1}^{k} f(x^\ell)$ the sum of the images of the first k sequence elements under f.

2.2 Job sequences

The released jobs form a discrete sequence, where at each time point the adversary releases at most one new job from every task. Formally, the adversary generates an infinite *job sequence* $\sigma = (\sigma^\ell)_{\ell \geqslant 1} \in \Sigma^\infty$, where $\Sigma = 2^{\mathcal{T}}$. The release of one job of task τ_i in time ℓ, for some $\ell \in \mathbb{N}^+$, is denoted by having $\tau_i \in \sigma^\ell$. Then, a (single) new job $J_{i,j}$ of task τ_i is released at the beginning of slot ℓ: $j = \ell$ denotes the *release time* of $J_{i,j}$, which is also the earliest time that the job $J_{i,j}$ can be executed, and $d_{i,j} = j + D_i$ denotes its absolute *deadline*.

2.3 Admissible job sequences

We present a flexible framework, where the set of admissible job sequences that the adversary can generate may be restricted. The set $\mathcal{J}$ of *admissible* job sequences from Σ^∞ can be obtained by imposing one or more of the following (optional) admissibility restrictions:

($\mathcal{S}$) Safety constraints, which are restrictions that have to hold in every finite prefix of an admissible job sequence; e.g., they can be used to enforce job release constraints such as periodicity or sporadicity, and to impose temporal workload restrictions.

($\mathcal{L}$) Liveness restrictions, which assert infinite repetition of certain job releases in a job sequence; e.g., they can be used to force the adversary to release a certain task infinitely often.

($\mathcal{W}$) Limit-average constraints, which restrict the long run average behavior of a job sequence; e.g., they can be used to enforce that the average load in the job sequences does not exceed a threshold.

These three types of constraints will be made precise in the next section where we formally state the problem definition.

2.4 Schedule

Given an admissible job sequence $\sigma \in \mathcal{J}$, the *schedule* $\pi = (\pi^\ell)_{\ell \geqslant 1} \in \Pi^\infty$, where $\Pi = ((\mathcal{T} \times \{0, \ldots, D_{\max} - 1\}) \cup \varnothing)$, computed by a real-time scheduling algorithm for σ, is a function that assigns at most one job for execution to every slot $\ell \geqslant 1$: π^ℓ is either $\varnothing$ (i.e., no job is executed) or else (τ_i, j) (i.e., the job $J_{i,\ell-j}$ of task τ_i released j slots ago is executed). The latter must satisfy the following constraints:

1. $\tau_i \in \sigma^{\ell-j}$ (the job has been released),
2. $j < D_i$ (the job's deadline has not passed),
3. $|\{k : k > 0 \text{ and } \pi^{\ell-k} = (\tau_i, j') \text{ and } k + j' = j\}| < C_i$ (the job released in slot $\ell - j$ has not been completed).

Note that our definition of schedules uses relative indexing in the scheduling algorithms: at time point ℓ, the scheduling algorithm computing π^ℓ uses index j to refer to slot $\ell - j$. Recall that $\pi(k)$ denotes the prefix of length $k \geqslant 1$ of π. We define $\gamma_i(\pi, k)$ to be the number of jobs of task τ_i that are completed by their deadlines in $\pi(k)$. The cumulated utility $V(\pi, k)$ (also called utility for brevity) achieved in $\pi(k)$ is defined as $V(\pi, k) = \sum_{i=1}^{N} \gamma_i(\pi, k) \cdot V_i$.

2.5 Competitive ratio

We are interested in evaluating the performance of deterministic *on-line* scheduling algorithms $\mathcal{A}$, which, at time ℓ, do not know any of the σ^k for $k > \ell$ when running on $\sigma \in \mathcal{J}$. In order to assess the performance of $\mathcal{A}$, we will compare the cumulated utility achieved in the schedule $\pi_{\mathcal{A}}$ to the cumulated utility achieved in the schedule $\pi_{\mathcal{C}}$ provided by an optimal *off-line* scheduling algorithm, called a *clairvoyant* algorithm $\mathcal{C}$, working on the same job sequence. Formally, given a taskset $\mathcal{T}$, let $\mathcal{J} \subseteq \Sigma^\infty$ be the set of all admissible job sequences of $\mathcal{T}$ that satisfy given (optional) safety, liveness, and limit-average constraints. For every $\sigma \in \mathcal{J}$, we denote with $\pi_{\mathcal{A}}^\sigma$ resp. $\pi_{\mathcal{C}}^\sigma$) the schedule produced by $\mathcal{A}$ (resp. $\mathcal{C}$) under σ. The *competitive ratio* of the on-line algorithm $\mathcal{A}$ for the taskset $\mathcal{T}$ under the admissible job sequence set $\mathcal{J}$ is defined as

$$\mathcal{CR}_{\mathcal{J}}(\mathcal{A}) = \inf_{\sigma \in \mathcal{J}} \liminf_{k \to \infty} \frac{1 + V(\pi^{\sigma}_{\mathcal{A}}, k)}{1 + V(\pi^{\sigma}_{\mathcal{C}}, k)} \tag{1}$$

that is, the worst-case ratio of the cumulated utility of the on-line algorithm versus the clairvoyant algorithm, under all admissible job sequences. Note that adding 1 in numerator and denominator simply avoids division by zero issues.

Remark 1 Since, according to the definition of the competitive ratio $\mathcal{CR}_{\mathcal{J}}$ in Eq. (1), we focus on worst-case analysis, we do not consider randomized algorithms (such as Locke's best-effort policy (Locke 1986)). Generally, for worst-case analysis, randomization can be handled by additional choices for the adversary. For the same reason, we do not consider scheduling algorithms that can use the unbounded history of job releases to predict the future (e.g., to capture correlations).

3 Modeling formalisms in our framework

In this section, we present the definitions of several types of labeled transition systems (LTSs). We use LTSs as the modeling formalism for on-line and clairvoyant scheduling algorithms, as well as for modeling optional constraints on the released job sequences.

3.1 Labeled Transition Systems

We will consider both on-line and off-line scheduling algorithms that are formally modeled as *labeled transition systems* (*LTSs*): every deterministic finite-state on-line scheduling algorithm can be represented as a deterministic LTS, such that every input job sequence generates a unique run that determines the corresponding schedule. On the other hand, an off-line algorithm can be represented as a non-deterministic LTS, which uses the non-determinism to guess the appropriate job to schedule.

3.1.1 Labeled transitions systems (LTSs)

Formally, a *labeled transition system* (LTS) is a tuple $L = (S, s_1, \Sigma, \Pi, \Delta)$, where S is a finite set of states, $s_1 \in S$ is the initial state, Σ is a finite set of input actions, Π is a finite set of output actions, and $\Delta \subseteq S \times \Sigma \times S \times \Pi$ is the transition relation. Intuitively, $(s, x, s', y) \in \Delta$ if, given the current state s and input x, the LTS outputs y and makes a transition to state s'. If the LTS is deterministic, then there is always a unique output and next state, i.e., Δ is a function $\Delta : S \times \Sigma \to S \times \Pi$. Given an input sequence $\sigma \in \Sigma^{\infty}$, a *run* of L on σ is a sequence $\rho_{\mathcal{A}} = (p_\ell, \sigma_\ell, q_\ell, \pi_\ell)_{\ell \geqslant 1} \in \Delta^{\infty}$ such that $p_1 = s_1$ and for all $\ell \geqslant 2$, we have $p_\ell = q_{\ell-1}$. For a deterministic LTS, for each input sequence, there is a unique run.

3.1.2 Deterministic LTS for an on-line algorithm

For our analysis, on-line scheduling algorithms are represented as deterministic LTSs. Recall the definition of the sets $\Sigma = 2^{\mathcal{T}}$, and $\Pi = ((\mathcal{T} \times \{0, \ldots, D_{\max} - 1\}) \cup$

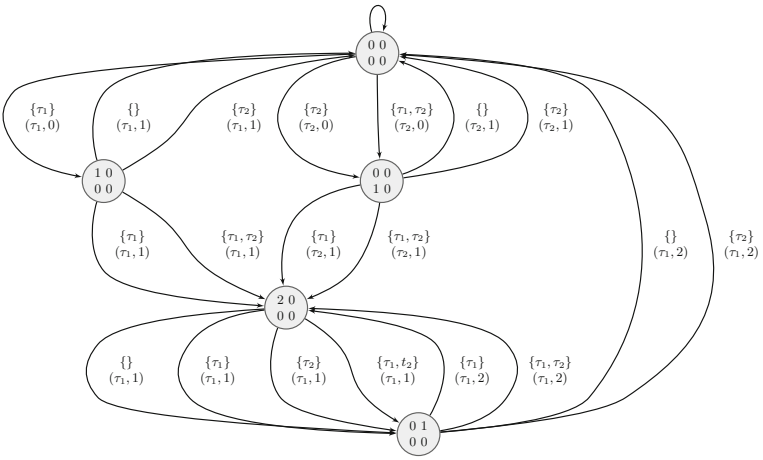


Fig. 1 EDF for $\mathcal{T} = \{\tau_1, \tau_2\}$ with $D_1 = 3$, $D_2 = 2$ and $C_1 = C_2 = 2$, represented as a deterministic LTS

$\varnothing$). Every deterministic on-line algorithm $\mathcal{A}$ that uses finite state space (for all job sequences) can be represented as a deterministic LTS $L_{\mathcal{A}} = (S_{\mathcal{A}}, s_{\mathcal{A}}, \Sigma, \Pi, \Delta_{\mathcal{A}})$, where the states $S_{\mathcal{A}}$ correspond to the state space of $\mathcal{A}$, and $\Delta_{\mathcal{A}}$ corresponds to the execution of $\mathcal{A}$ for one slot. Note that, due to relative indexing, for every current slot ℓ, the schedule $(\pi^\ell)_{\ell \geqslant 1}$ of $\mathcal{A}$ contains elements π^ℓ from the set Π, and $\pi^\ell = (\tau_i, j)$ uniquely determines the job $J_{i,\ell-j}$. Finally, we associate with $L_{\mathcal{A}}$ a reward function $r_{\mathcal{A}} : \Delta_{\mathcal{A}} \to \mathbb{N}$ such that $r_{\mathcal{A}}(\delta) = V_i$ if the transition δ completes a job of task τ_i, and $r_{\mathcal{A}}(\delta) = 0$ otherwise. Given the unique run $\rho^\sigma_{\mathcal{A}} = (\delta^\ell)_{\ell \geqslant 1}$ of $L_{\mathcal{A}}$ for the job sequence σ, where δ^ℓ denotes the transition taken at the beginning of slot ℓ, the cumulated utility in the prefix of the first k transitions in $\rho^\sigma_{\mathcal{A}}$ is $V(\rho^\sigma_{\mathcal{A}}, k) = \sum_{\ell=1}^{k} r_{\mathcal{A}}(\delta^\ell)$.

Most scheduling algorithms [such as EDF, FIFO, DOVER (Koren and Shasha 1995), TD1 (Baruah et al. 1992)] can be represented as a deterministic LTS. An illustration for EDF is given in the following example.

Example 1 Consider the taskset $\mathcal{T} = \{\tau_1, \tau_2\}$, with $D_1 = 3$, $D_2 = 2$ and $C_1 = C_2 = 2$. Figure 1 represents the EDF (Earliest Deadline First) scheduling policy as a deterministic LTS for $\mathcal{T}$. Each state is represented by a matrix M, such that $M[i, j]$, $1 \leqslant i \leqslant N, 1 \leqslant j \leqslant D_{\max} - 1$, denotes the remaining execution time of the job of task τ_i released j slots ago. Every transition is labeled with a set $T \in \Sigma$ of released tasks as well as with $(\tau_i, j) \in \Pi$, which denotes the unique job $J_{i,\ell-j}$ to be scheduled in the current slot ℓ. Released jobs with no chance of being scheduled are not included in the state space. For example, while being in the topmost state, the release of τ_1 makes the LTS take the transition to the leftmost state, where 1 unit of work is scheduled for the released task, and 1 unit remains, encoded by writing 1 in position (1, 1) of the matrix M. In the next round, a new release of τ_2 will take the LTS to the middle state, with 2 units of workload in position (1, 1). This is because the 2nd workload of the first job is scheduled (thus the first job is scheduled to completion), and the newly released job is not scheduled in the current slot. Thus all 2 units of workload of the currently released job remain.

All scheduling algorithms considered in this paper have been encoded similarly to EDF using the matrix M. Some more involved schedulers, such as DOVER, require some extra bits of information stored in the state.

3.1.3 The non-deterministic LTS

The clairvoyant algorithm $\mathcal{C}$ is formally a non-deterministic LTS $L_{\mathcal{C}} = (S_{\mathcal{C}}, s_{\mathcal{C}}, \Sigma, \Pi, \Delta_{\mathcal{C}})$, where each state in $S_{\mathcal{C}}$ is a $N \times (D_{\max} - 1)$ matrix M. For each time slot ℓ, the entry $M[i, j]$, $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant D_{\max} - 1$, denotes the remaining execution time of the job $J_{i,\ell-j}$ (i.e., the job of task i released j slots ago). For matrices M, M', subset $T \in \Sigma$ of newly released tasks, and scheduled job $P = (\tau_i, j) \in \Pi$, we have $(M, T, M', P) \in \Delta_{\mathcal{C}}$ iff $M[i, j] > 0$ and M' is obtained from M by

1. inserting all $\tau_i \in T$ into column 1 of M,
2. decrementing the value at position $M[i, j]$, and
3. shifting the contents of M by one column to the right, while initializing column 1 to all 0.

That is, M' corresponds to M after inserting all released tasks in the current state, executing a pending task for one unit of time, and reducing the relative deadlines of all tasks currently in the system. The initial state $s_{\mathcal{C}}$ is represented by the zero $N \times (D_{\max} - 1)$ matrix, and $S_{\mathcal{C}}$ is the smallest $\Delta_{\mathcal{C}}$-closed set of states that contains $s_{\mathcal{C}}$ (i.e., if $M \in S_{\mathcal{C}}$ and $(M, T, M', P) \in \Delta_{\mathcal{C}}$ for some T, M' and P, we have $M' \in S_{\mathcal{C}}$). Finally, we associate with $L_{\mathcal{C}}$ a reward function $r_{\mathcal{C}} : \Delta_{\mathcal{C}} \to \mathbb{N}$ such that $r_{\mathcal{C}}(\delta) = V_i$ if the transition δ completes a task τ_i, and $r_{\mathcal{C}}(\delta) = 0$ otherwise.

Remark 2 Note that the size of the above LTSs is the size of the state space of the corresponding scheduling algorithm. If the input consists of a succinct description of these algorithms [e.g., as a circuit (Galperin and Wigderson 1983)], then the size of the corresponding LTS is, in general, exponential in the size of the input. This state-space explosion is generally unavoidable (Clarke et al. 1999). In the complexity analysis of our algorithms, we consider all scheduling algorithms to be given in the explicit form of LTSs. When appropriate, we will state what the obtained results imply for the case where the input is succinct.

3.2 Admissible job sequences

Our framework allows to restrict the adversary to generate admissible job sequences $\mathcal{J} \subseteq \Sigma^{\infty}$, which can be specified via different constraints. Since a constraint on job sequences can be interpreted as a language (which is a subset of infinite words Σ^{∞} here), we will use automata as acceptors of such languages. Since an automaton is a deterministic LTS with no output, all our constraints will be described as LTSs with an empty set of output actions. We allow the following types of constraints:

($\mathcal{S}$) Safety constraints are defined by a deterministic LTS $L_{\mathcal{S}} = (S_{\mathcal{S}}, s_{\mathcal{S}}, \Sigma, \varnothing, \Delta_{\mathcal{S}})$, with a distinguished *absorbing* reject state $s_r \in S_{\mathcal{S}}$. An absorbing state is a state that has outgoing transitions only to itself. Every job sequence σ defines

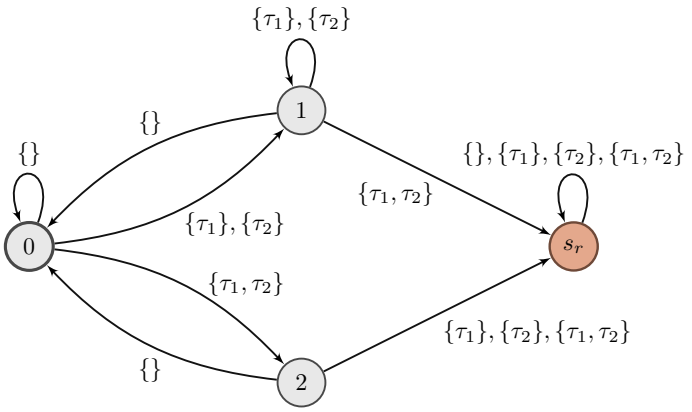


Fig. 2 Example of a safety LTS $L_{\mathcal{S}}$ that restricts the adversary to release at most 2 units of workload in the last 2 slots. In state 0, no workload has been released in the last 2 slots, and thus all task releases are allowed for the next time slot. In state 1, there has been 1 unit of workload released in the last 2 slots, and thus in the next slot only one task can be released. If no task is released in the next slot, then we transition back to state 0, to indicate that in the next time slot any combination of task releases is allowed. In state 2, there have been 2 units of workload released in the last 2 slots, and thus no task release is allowed in the next slot. If no tasks are released, then the LTS transitions back to state 0, as in the next time slot any combination of task releases is allowed. If any of the above rules is violated, the safety LTS transitions to the absorbing state s_r, and remains there forever to indicate that the workload restriction has been violated

a unique run $\rho^{\sigma}_{\mathcal{S}}$ in $L_{\mathcal{S}}$, such that either no transition to s_r appears in $\rho^{\sigma}_{\mathcal{S}}$, or every such transition is followed solely by self-transitions to s_r. A job sequence σ is *admissible* to $L_{\mathcal{S}}$, if $\rho^{\sigma}_{\mathcal{S}}$ does not contain a transition to s_r. To obtain a safety LTS that does not restrict $\mathcal{J}$ at all, we simply use a trivial deterministic $L_{\mathcal{S}}$ with no transition to s_r. Safety constraints restrict the adversary to release job sequences, where every finite prefix satisfies some property (as they lead to the absorbing reject state s_r of $L_{\mathcal{S}}$ otherwise). Some well-known examples of safety constraints are (i) periodicity and/or sporadicity constraints, where there are fixed and/or a minimum time between the release of any two consecutive jobs of a given task, and (ii) absolute workload constraints (Golestani 1991; Cruz 1991), where the total workload released in the last k slots, for some fixed k, is not allowed to exceed a threshold λ. For example, in the case of absolute workload constraints, $L_{\mathcal{S}}$ simply encodes the workload in the last k slots in its state, and makes a transition to s_r whenever the workload exceeds λ. Figure 2 shows an example of a constraint LTS for the taskset $\mathcal{T} = \{\tau_1, \tau_2\}$ with $C_1 = C_2 = 1$ that restricts the adversary to release at most 2 units of workload in the last 2 slots.

($\boldsymbol{\mathcal{L}}$) Liveness constraints are modeled as a deterministic LTS $L_{\mathcal{L}} = (S_{\mathcal{L}}, s_{\mathcal{L}}, \Sigma, \varnothing, \Delta_{\mathcal{L}})$ with a distinguished *accept* state $s_a \in S_{\mathcal{L}}$. A job sequence σ is *admissible* to the liveness LTS $L_{\mathcal{L}}$ if $\rho^{\sigma}_{\mathcal{L}}$ contains infinitely many transitions to s_a. For the case where there are no liveness constraint in $\mathcal{J}$, we use a LTS $L_{\mathcal{L}}$ consisting of state s_a only. Liveness constraints force the adversary to release job sequences that satisfy some property infinitely often. For example, they could be used to guarantee that the release of some particular task τ_i does not eventually stall;

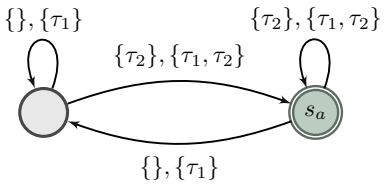


Fig. 3 Example of a liveness LTS $L_{\mathcal{L}}$ that forces τ_2 to be released infinitely often. Each time the τ_2 is released, the LTS transitions to the accepting state s_a to indicate the release of the desired task. Recall that the accepting condition of $L_{\mathcal{L}}$ is that s_a needs to be appear infinitely often in an accepting path, meaning that the task τ_2 appears infinitely often

the constraint is specified by a two-state LTS $L_{\mathcal{L}}$ that visits s_a whenever the current job set includes τ_i. A liveness constraint can also be used to prohibit infinitely long periods of overload (Baruah et al. 1992), by choosing s_a as the idle state. Figure 3 shows an example of a constraint LTS for the taskset $\mathcal{T} = \{\tau_1, \tau_2\}$ that forces the adversary to release τ_2 infinitely often.

($\mathcal{W}$) Limit-average constraints are defined by a deterministic weighted LTS $L_{\mathcal{W}} = (S_{\mathcal{W}}, s_{\mathcal{W}}, \Sigma, \varnothing, \Delta_{\mathcal{W}})$ equipped with a weight function $w : \Delta_{\mathcal{W}} \to \mathbb{Z}^d$ that assigns a vector of weights to every transition $\delta_{\mathcal{W}} \in \Delta_{\mathcal{W}}$. Given a threshold vector $\vec{\lambda} \in \mathbb{Q}^d$, where $\mathbb{Q}$ denotes the set of all rational numbers, a job sequence σ and the corresponding run $\rho^{\sigma}_{\mathcal{W}} = (\delta^{\ell}_{\mathcal{W}})_{\ell \geqslant 1}$ of $L_{\mathcal{W}}$, the job sequence is *admissible* to $L_{\mathcal{W}}$ if $\liminf_{k\to\infty} \frac{1}{k} \cdot w(\rho^{\sigma}_{\mathcal{W}}, k) \leqslant \vec{\lambda}$ with $w(\rho^{\sigma}_{\mathcal{W}}, k) = \sum_{i=1}^{k} w(\delta^{\ell}_{\mathcal{W}})$.
Consider a relaxed notion of workload constraints, where the adversary is restricted to generate job sequences whose *average* workload does not exceed a threshold λ. Since this constraint still allows "busy" intervals where the workload temporarily exceeds λ, it cannot be expressed as a safety constraint. To support such interesting average constraints of admissible job sequences, where the adversary is more relaxed than under absolute constraints, our framework explicitly supports limit-average constraints. Therefore, it is possible to express the average workload assumptions commonly used in the analysis of aperiodic task scheduling in soft-real-time systems (Abeni and Buttazzo 1998; Haritsa et al. 1990). Other interesting cases of limit-average constraints include restricting the average sporadicity, and, in particular, average energy: ensuring that the limit-average of the energy consumption is below a certain threshold is an important concern in modern real-time systems (Aydin et al. 2004). Figure 4 shows an example of a constraint LTS for the taskset $\mathcal{T} = \{\tau_1, \tau_2\}$ with $C_1 = C_2 = 1$, which can be used to restrict the average workload the adversary is allowed to release in the long run.

Remark 3 While, in general, such constraints are encoded as independent automata, it is often possible to encode certain constraints directly in the non-deterministic LTS of the clairvoyant scheduler instead. In particular, this is possible for restricting the limit-average workload, generating finite intervals of overload, and releasing a particular job infinitely often.

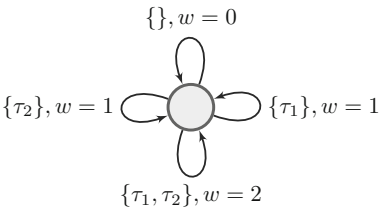


Fig. 4 Example of a limit-average LTS $L_{\mathcal{W}}$ that tracks the average workload of jobs released by the adversary. This is achieved by having the weight function indicate the total workload released in each time slot. In this case we have $C_1 = C_2 = 1$, and the total workload equals the number of released tasks

3.2.1 Synchronous product of LTSs

The *synchronous product* of two LTSs $L_1 = (S_1, s_1, \Sigma, \Pi, \Delta_1)$ and $L_2 = (S_2, s_2, \Sigma, \Pi, \Delta_2)$ is an LTS $L = (S, s, \Sigma, \Pi', \Delta)$ such that:

1. $S \subseteq S_1 \times S_2$,
2. $s = (s_1, s_2)$,
3. $\Pi' = \Pi \times \Pi$, and
4. $\Delta \subseteq S \times \Sigma \times S \times \Pi'$ such that $((q_1, q_2), T, (q_1', q_2'), (P_1, P_2)) \in \Delta$ iff $(q_1, T, q_1', P_1) \in \Delta_1$ and $(q_2, T, q_2', P_2) \in \Delta_2$.

The set of states S is the smallest Δ-closed subset of $S_1 \times S_2$ that contains s (i.e., $s \in S$, and for each $q \in S$, if there exist $q' \in S_1 \times S_2$, $T \in \Sigma$ and $P \in \Pi'$ such that $(q, T, q', P) \in \Delta$, then $q' \in S$). That is, the synchronous product of L_1 with L_2 captures the joint behavior of L_1 and L_2 in every input sequence $\sigma \in \Sigma^\infty$ (L_1 and L_2 synchronize on input actions). Note that if both L_1 and L_2 are deterministic, so is their synchronous product. The synchronous product of $k > 2$ LTSs $L_1, \ldots, L_k$ is defined iteratively as the synchronous product of L_1 with the synchronous product of $L_2, \ldots, L_k$.

3.2.2 Overall approach for computing $\mathcal{CR}$

Our goal is to determine the worst-case competitive ratio $\mathcal{CR}_{\mathcal{J}}(\mathcal{A})$ for a given on-line algorithm $\mathcal{A}$. The inputs to the problem are the given taskset $\mathcal{T}$, an on-line algorithm $\mathcal{A}$ specified as a deterministic LTS $L_{\mathcal{A}}$, and the safety, liveness, and limit-average constraints specified as deterministic LTSs $L_{\mathcal{S}}$, $L_{\mathcal{L}}$ and $L_{\mathcal{W}}$, respectively, which constrain the admissible job sequences $\mathcal{J}$. Our approach uses a reduction to a multi-objective graph problem, which consists of the following steps:

1. Construct a non-deterministic LTS $L_{\mathcal{C}}$ corresponding to the clairvoyant off-line algorithm $\mathcal{C}$. Note that since $L_{\mathcal{C}}$ is non-deterministic, for every admissible job sequence σ, there are many possible runs in $L_{\mathcal{C}}$, of course also including the runs with maximum cumulative utility.
2. Take the synchronous product LTS $L_{\mathcal{A}} \times L_{\mathcal{C}} \times L_{\mathcal{S}} \times L_{\mathcal{L}} \times L_{\mathcal{W}}$. By doing so, a path in the product graph corresponds to *identically* labeled paths in the LTSs,

and thus ensures that they agree on the same job sequence σ. This product can be represented by a multi-objective graph (as introduced in Sect. 4.1).
3. Determine $\mathcal{CR}_{\mathcal{J}}(\mathcal{A})$ by reducing the computation of the ratio given in Eq. (1) to solving a multi-objective problem on the product graph.
4. Employ several optimizations in order to reduce the size of product graph (see Sects. 4.3 and 4.4).

4 Competitive analysis of on-line scheduling algorithms

In this section, we address the competitive analysis problem: given a taskset, a LTS $L_{\mathcal{A}}$ for the on-line scheduling algorithm, and optional constraint automata $L_{\mathcal{S}}$, $L_{\mathcal{L}}$, $L_{\mathcal{W}}$ for the set of admissible job sequences $\mathcal{J}$, our algorithms compute the competitive ratio $\mathcal{CR}_{\mathcal{J}}(\mathcal{A})$ of $\mathcal{A}$ in $\mathcal{J}$. Our presentation is organized as follows: in Sect. 4.1, we define qualitative and quantitative objectives on multi-graphs. In Sect. 4.2, we provide algorithms for solving these graph objectives. In Sect. 4.3, we establish a formal reduction of computing the competitive ratio $\mathcal{CR}_{\mathcal{J}}(\mathcal{A})$ of an on-line scheduling algorithm $\mathcal{A}$ to solving for graph objectives on a suitable multi-graph. In Sect. 4.4, we describe several generic optimizations for this reduction that make the reduction practical. In Sect. 4.5, we provide the results of an automatic competitive analysis of a wide range of classic on-line scheduling algorithms, using a prototype implementation of our framework.

4.1 Graphs with multiple objectives

In this subsection, we define various objectives on graphs and outline the algorithms for solving them. We later show how the competitive analysis of on-line schedulers reduces to the solution algorithms of this section.

4.1.1 Multi-graphs

A *multi-graph* $G = (V, E)$, hereinafter called simply a *graph*, consists of a finite set V of n *nodes*, and a finite set of m *directed multiple edges* $E \subset V \times V \times \mathbb{N}^+$. For brevity, we will refer to an edge (u, v, i) as (u, v), when i is not relevant. We consider graphs in which for all $u \in V$, we have $(u, v) \in E$ for some $v \in V$, i.e., every node has at least one outgoing edge. An *infinite path* ρ of G is an infinite sequence of edges $e^1, e^2, \ldots$ such that, for all $i \geqslant 1$ with $e^i = (u^i, v^i)$, we have $v^i = u^{i+1}$. Every such path ρ induces a sequence of nodes $(u^i)_{i \geqslant 1}$, which we will also call a path when the distinction is clear from the context, where ρ^i refers to u^i instead of e^i. Finally, we denote by Ω the set of all paths of G.

4.1.2 Objectives

Given a graph G, an objective Φ is a subset of Ω that defines the desired set of paths. We will consider safety, liveness, mean-payoff (limit-average), and ratio objectives, and their conjunction for multiple objectives.

Safety and liveness objectives We consider safety and liveness objectives, both defined with respect to some subset of nodes $X, Y \subseteq V$. Given $X \subseteq V$, the *safety* objective, defined as $\mathsf{Safe}(X) = \{\rho \in \Omega : \forall i \geqslant 1, \rho^i \notin X\}$, represents the set of all paths that never visit the set X. The *liveness* objective defined as $\mathsf{Live}(Y) = \{\rho \in \Omega : \forall j \exists i > j \text{ s.t. } \rho^i \in Y\}$ represents the set of all paths that visit Y infinitely often.
Mean-payoff and ratio objectives We consider the mean-payoff and ratio objectives, defined with respect to a weight function and a threshold. A *weight function* $w : E \to \mathbb{Z}^d$ assigns to each edge of G a vector of d integers. A weight function naturally extends to paths, with $w(\rho, k) = \sum_{i=1}^{k} w(\rho^i)$.

The *mean-payoff* (or limit-average) of a path ρ is defined as:

$$\mathsf{MP}(w, \rho) = \liminf_{k \to \infty} \frac{1}{k} \cdot w(\rho, k);$$

i.e., it is the long-run average of the weights of the path. Given a weight function w and a threshold vector $\vec{\nu} \in \mathbb{Q}^d$, the corresponding objective is given as:

$$\mathsf{MP}(w, \vec{\nu}) = \{\rho \in \Omega : \mathsf{MP}(w, \rho) \leqslant \vec{\nu}\};$$

that is, the set of all paths such that the mean-payoff of their weights is at most $\vec{\nu}$ (where we consider pointwise comparison for vectors). For weight functions w_1, $w_2 : E \to \mathbb{N}^d$, the *ratio* of a path ρ is defined as:

$$\mathsf{Ratio}(w_1, w_2, \rho) = \liminf_{k \to \infty} \frac{\vec{\mathbf{1}} + w_1(\rho, k)}{\vec{\mathbf{1}} + w_2(\rho, k)},$$

which denotes the limit infimum of the coordinate-wise ratio of the sum of weights of the two functions; $\vec{\mathbf{1}}$ denotes the d-dimensional all-1 vector. Given weight functions w_1, w_2 and a threshold vector $\vec{\nu} \in \mathbb{Q}^d$, the ratio objective is given as:

$$\mathsf{Ratio}(w_1, w_2, \vec{\nu}) = \{\rho \in \Omega : \mathsf{Ratio}(w_1, w_2, \rho) \leqslant \vec{\nu}\}$$

that is, the set of all paths such that the ratio of cumulative rewards w.r.t w_1 and w_2 is at most $\vec{\nu}$.

Example 2 Consider the multi-graph shown in Fig. 5, with a weight function of dimension $d = 2$. Note that there are two edges from node 3 to node 5 (represented as edges $(3, 5, 1)$ and $(3, 5, 2)$). In the graph we have a weight function with dimension 2. Note that the two edges from node 3 to node 5 have incomparable weight vectors.

4.1.3 Decision problem

The decision problem we consider is as follows: given the graph G, an initial node $s \in V$, and an objective Φ (which can be a conjunction of several objectives), determine whether there exists a path ρ that starts from s and belongs to Φ, i.e., $\rho \in \Phi$. For

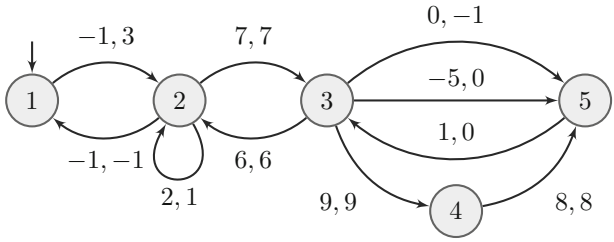


Fig. 5 An example of a multi-graph G

simplicity of presentation, we assume that every $u \in V$ is reachable from s (unreachable nodes can be discarded by preprocessing G in $\mathcal{O}(m)$ time). We first present algorithms for each of safety, liveness, mean-payoff, and ratio objectives separately, and then for their conjunction.

4.2 Algorithms for solving graphs with multiple objectives

We now describe the algorithms for solving the graph objectives introduced in the last subsection.

4.2.1 Algorithms for safety and liveness objectives

The algorithm for the objective $\mathsf{Safe}(X)$ is straightforward. We first remove the set X of nodes and then perform an SCC (maximal strongly connected component) decomposition of G. Then, we perform a single graph traversal to identify the set of nodes V_X which can reach an SCC that contains at least one edge (i.e., it contains either a single node with a self-loop, or more than one nodes). Note that since we have removed the set X, we have that $V_X \cap X = \varnothing$. In the end, we obtain a graph $G = (V_X, E_X)$ such that $E_X = E \cap (V_X \times V_X)$. Thus, the objective $\mathsf{Safe}(X)$ is satisfied in the resulting graph, and the algorithm answers yes iff $s \in V_X$. Using the algorithm of Tarjan (1972) for performing the SCC decomposition, this algorithm requires $\mathcal{O}(m)$ time.

To solve for the objective $\mathsf{Live}(Y)$, initially perform an SCC decomposition of G. We call an SCC V_{SCC} *live*, if (i) either $|V_{\mathrm{SCC}}| > 1$, or $V_{\mathrm{SCC}} = \{u\}$ and $(u, u) \in E$; and (ii) $V_{\mathrm{SCC}} \cap Y \neq \varnothing$. Then $\mathsf{Live}(Y)$ is satisfied in G iff there exists a live SCC V_{SCC} that is reachable from s. This is because every node u in V_{SCC} can reach every node in V_{SCC}, and thus there is a path $u \rightsquigarrow u$ in V_{SCC}. Since V_{SCC} is a live SCC, the same holds for nodes $u \in V_{\mathrm{SCC}} \cap Y$. Then a witness path can be constructed which first reaches some node $u \in V_{\mathrm{SCC}} \cap Y$, and then keeps repeating the path $u \rightsquigarrow u$. Using the algorithm of (Tarjan 1972) for performing the SCC decomposition, this algorithm also requires $\mathcal{O}(m)$ time.

4.2.2 Algorithms for mean-payoff objectives

We distinguish between the case when the weight function has a single dimension ($d = 1$) versus the case when the weight function has multiple dimensions ($d > 1$).

Single dimension In the case of a single-dimensional weight function, a single weight is assigned to every edge, and the decision problem of the mean-payoff objective reduces to determining the mean weight of a minimum-weight simple cycle in G, as the latter also determines the mean-weight by infinite repetition. Using the algorithms of Karp (1978 and Madani (2002), this process requires $\mathcal{O}(n \cdot m)$ time. When the objective is satisfied, the process also returns a simple cycle C, as a witness to the objective. From C, a path $\rho \in \mathsf{MP}(w, \vec{\nu})$ is constructed by infinite repetitions of C.

Multiple dimensions When $d > 1$, the mean-payoff objective reduces to determining the feasibility of a linear program (LP). For $u \in V$, let $\mathsf{IN}(u)$ be the set of incoming, and $\mathsf{OUT}(u)$ the set of outgoing edges of u. As shown in Velner et al. (2015), G satisfies $\mathsf{MP}(w, \vec{\nu})$ iff the following set of constraints on $\vec{x} = (x_e)_{e \in E_{\mathrm{SCC}}}$ with $x_e \in \mathbb{Q}$ is satisfied simultaneously on some SCC V_{SCC} of G with induced edges $E_{\mathrm{SCC}} \subseteq E$.

$$
\begin{aligned}
& x_e \geqslant 0 && e \in E_{\mathrm{SCC}} \\
& \sum_{e \in \mathsf{IN}(u)} x_e = \sum_{e \in \mathsf{OUT}(u)} x_e && u \in V_{\mathrm{SCC}} \\
& \sum_{e \in E_{\mathrm{SCC}}} x_e \cdot w(e) \leqslant \vec{\nu} \\
& \sum_{e \in E_{\mathrm{SCC}}} x_e \geqslant 1
\end{aligned}
\tag{2}
$$

The quantities x_e are intuitively interpreted as "flows". The first constraint specifies that the flow of each edge is non-negative. The second constraint is a flow-conservation constraint. The third constraint specifies that the objective is satisfied if we consider the relative contribution of the weight of each edge, according to the flow of the edge. The last constraint asks that the preceding constraints are satisfied by a non-trivial (positive) flow. Hence, when $d > 1$, the decision problem reduces to solving a LP, and the time complexity is polynomial (Khachiyan 1979).

The witness path construction from a feasible solution consists of two steps:

1. Construction of a multi-cycle from the feasible solution; and
2. Construction of an infinite witness path from the multi-cycle.

We describe the two steps in detail. Formally, a *multi-cycle* is a finite set of cycles with multiplicity $\mathcal{MC} = \{(C_1, m_1), (C_2, m_2), \ldots, (C_k, m_k)\}$, such that every C_i is a simple cycle and m_i is its multiplicity. The construction of a multi-cycle from a feasible solution $\vec{x}$ is as follows. Let $\mathcal{E} = \{e : x_e > 0\}$. By scaling each edge flow x_e by a common factor z, we construct the set $\mathcal{X} = \{(e, z \cdot x_e) : e \in \mathcal{E}\}$, with $\mathcal{X} \subset E_{\mathrm{SCC}} \times \mathbb{N}^+$. Then, we start with $\mathcal{MC} = \varnothing$ and apply iteratively the following procedure until $\mathcal{X} = \varnothing$:

(i) find a pair $(e_i, m_i) = \arg\min_{(e_j, m_j) \in \mathcal{X}} m_j$,
(ii) form a cycle C_i that contains e_i and only edges that appear in $\mathcal{X}$ (because of Eq. (2), this is always possible),

(iii) add the pair (C_i, m_i) in the multi-cycle $\mathcal{MC}$,
(iv) subtract m_i from all elements (e_j, m_j) of $\mathcal{X}$ such that the edge e_j appears in C_i,
(v) remove from $\mathcal{X}$ all $(e_j, 0)$ pairs, and repeat.

Since V_{SCC} is an SCC, there is a path $C_i \rightsquigarrow C_j$ for all C_i, C_j in $\mathcal{MC}$. Given the multi-cycle $\mathcal{MC}$, the infinite path that achieves the weight at most $\vec{\nu}$ is not periodic, but generated by Algorithm 1. Note that perpetually increasing ℓ in Line 9 ensures that the contributions of the (finite) intermediate paths $C_1 \rightsquigarrow C_2$, etc. vanish in the limit.

Algorithm 1: Multi-objective witness

Input: A graph $G = (V, E)$, and a multi-cycle $\mathcal{MC} = \{(C_1, m_1), (C_2, m_2), \ldots, (C_k, m_k)\}$
Output: An infinite path $\rho \in \mathsf{MP}(w, \vec{\nu})$

```
1  ℓ ← 1
2  while True do
3      Repeat C_1 for ℓ · m_1 times
4      C_1 ⇝ C_2
5      Repeat C_2 for ℓ · m_2 times
6      ...
7      Repeat C_k for ℓ · m_k times
8      C_k ⇝ C_1
9      ℓ ← ℓ + 1
10 end
```

4.2.3 Algorithm for ratio objectives

We now consider ratio objectives, and present a reduction to mean-payoff objectives. Consider the weight functions w_1, w_2 and the threshold vector $\vec{\nu} = \frac{\vec{p}}{\vec{q}}$ as the component-wise division of vectors $\vec{p}, \vec{q} \in \mathbb{N}^d$. We define a new weight function $w : E \to \mathbb{Z}^d$ such that, for all $e \in E$, we have $w(e) = \vec{q} \cdot w_1(e) - \vec{p} \cdot w_2(e)$ (where $\cdot$ denotes component-wise multiplication). It is easy to verify that $\mathsf{Ratio}(w_1, w_2, \vec{\nu}) = \mathsf{MP}(w, \vec{0})$, and thus we solve the ratio objective by solving the new mean-payoff objective, as described above.

4.2.4 Algorithms for conjunctions of objectives

Finally, we consider the conjunction of a safety, a liveness, and a mean-payoff objective (note that we have already described a reduction of ratio objectives to mean-payoff objectives). More specifically, given a weight function w, a threshold vector $\vec{\nu} \in \mathbb{Q}$, and sets $X, Y \subseteq V$, we consider the decision problem for the objective $\Phi = \mathsf{Safe}(X) \cap \mathsf{Live}(Y) \cap \mathsf{MP}(w, \vec{\nu})$. The procedure is as follows:

1. Initially compute G_X from G as in the case of a single safety objective.
2. Then, perform an SCC decomposition on G_X.

3. For every live SCC V_{SCC} that is reachable from s, solve for the mean-payoff objective in V_{SCC}. Return yes, if $\mathsf{MP}(w, \vec{\nu})$ is satisfied in any such V_{SCC}.

If the answer to the decision problem is yes, then the witness consists of a live SCC V_{SCC}, along with a multi-cycle (resp. a cycle for $d = 1$). The witness infinite path is constructed as in Algorithm 1, with the only difference that at the end of each while loop a live node from Y in the SCC V_{SCC} is additionally visited. The time required for the conjunction of objectives is dominated by the time required to solve for the mean-payoff objective. Theorem 1 summarizes the results of this section.

Theorem 1 *Let $G = (V, E)$ be a graph, $s \in V$, $X, Y \subseteq V$, $w : E \to \mathbb{Z}^d$, w_1, w_2: $E \to \mathbb{N}^d$ weight functions, and $\vec{\nu} \in \mathbb{Q}^d$. Let $\Phi_1 = \mathsf{Safe}(X) \cap \mathsf{Live}(Y) \cap \mathsf{MP}(w, \vec{\nu})$ and $\Phi_2 = \mathsf{Safe}(X) \cap \mathsf{Live}(Y) \cap \mathsf{Ratio}(w_1, w_2, \vec{\nu})$. The decision problem of whether G satisfies the objective Φ_1 (resp. Φ_2) from s requires*

1. *$\mathcal{O}(n \cdot m)$ time, if $d = 1$.*
2. *Polynomial time, if $d > 1$.*

If the objective Φ_1 (resp. Φ_2) is satisfied in G from s, then a finite witness (an SCC and a cycle for single dimension, and an SCC and a multi-cycle for multiple dimensions) exists and can be constructed in polynomial time.

Example 3 Consider the graph in Fig. 5. Starting from node 1, the mean-payoff-objective $\mathsf{MP}(w, \vec{0})$ is satisfied by the multi-cycle $\mathcal{MC} = \{(C_1, 1), (C_2, 2)\}$, with $C_1 = [(1, 2), (2, 1)]$ and $C_2 = [(3, 5), (5, 3)]$. A solution to the corresponding LP is $x_{(1,2)} = x_{(2,1)} = \frac{1}{3}$ and $x_{(3,5)} = x_{(5,3)} = \frac{2}{3}$, and $x_e = 0$ for all other $e \in E$. Procedure 1 then generates a witness path for the objective. The objective is also satisfied in conjunction with $\mathsf{Safe}(\{4\})$ or $\mathsf{Live}(\{4\})$. In the latter case, a witness path additionally traverses the edges $(3, 4)$ and $(4, 5)$ before transitioning from C_1 to C_2.

Example 4 Consider the same graph of Fig. 5, where now instead of a single weight function of two dimensions, we have two weight functions $w_1, w_2 : E \to \mathbb{Z}$, of a single dimension each. The first (resp. second) weight of each edge is with respect to the weight function w_1 (resp. w_2). The ratio objective $\mathsf{Ratio}(w_1, w_2, -4)$ is satisfied by traversing the cycle $C = [(3, 5), (5, 3)]$ repeatedly.

4.3 Reduction of competitive analysis to graphs with multiple objectives

We present a formal reduction of the computation of the competitive ratio of an online scheduling algorithm with constraints on job sequences to the multi-objective graph problem. The input consists of the taskset, a deterministic LTS for the online algorithm, a non-deterministic LTS for the clairvoyant algorithm, and optional deterministic LTSs for the constraints. We first describe the process of computing the competitive ratio $\mathcal{CR}_{\mathcal{J}}(\mathcal{A})$, where $\mathcal{J}$ is a set of job sequences only subject to safety and liveness constraints. We later show how to handle limit-average constraints.

4.3.1 Reduction for safety and liveness constraints

Given the deterministic and non-deterministic LTS $L_{\mathcal{A}}$ and $L_{\mathcal{C}}$ with reward functions $r_{\mathcal{A}}$ and $r_{\mathcal{C}}$, respectively, and optionally safety and liveness LTS $L_{\mathcal{S}}$ and $L_{\mathcal{L}}$, let $L =$

$L_{\mathcal{A}} \times L_{\mathcal{C}} \times L_{\mathcal{S}} \times L_{\mathcal{L}}$ be their synchronous product. Hence, L is a non-deterministic LTS $(S, s_1, \Sigma, \Pi, \Delta)$, and every job sequence σ yields a set of runs R in L, such that each $\rho \in R$ captures the joint behavior of $\mathcal{A}$ and $\mathcal{C}$ under σ. Note that for each such ρ the behavior of $\mathcal{A}$ is unchanged, but the behavior of $\mathcal{C}$ generally varies due to its non-determinism. Let $G = (V, E)$ be the multi-graph induced by L, that is, $V = S$ and $(M, M', j) \in E$ for all $1 \leqslant j \leqslant i$ iff there are i transitions $(M, T, M', P) \in \Delta$. Let $w_{\mathcal{A}}$ and $w_{\mathcal{C}}$ be the weight functions that assign to each edge of G the reward that the respective algorithm obtains from the corresponding transition in L. Let X be the set of states in G whose $L_{\mathcal{S}}$ component is s_r, and Y the set of states in G whose $L_{\mathcal{L}}$ component is s_a. It follows that for all $\nu \in \mathbb{Q}$, we have that $\mathcal{CR}_{\mathcal{J}}(\mathcal{A}) \leqslant \nu$ iff the objective $\Phi_\nu = \mathsf{Safe}(X) \cap \mathsf{Live}(Y) \cap \mathsf{Ratio}(w_{\mathcal{A}}, w_{\mathcal{C}}, \nu)$ is satisfied in G from the state s_1. As the dimension in the ratio objective (that just takes care of the competitive ratio) is one, Case 1 of Theorem 1 applies, and we obtain the following:

Lemma 1 *Given the product graph $G = (V, E)$ of n nodes and m edges, a rational $\nu \in \mathbb{Q}$, and a set of job sequences $\mathcal{J}$ admissible for safety and liveness LTSs, determining whether $\mathcal{CR}_{\mathcal{J}}(\mathcal{A}) \leqslant \nu$ requires $O(n \cdot m)$ time.*

Since $0 \leqslant \mathcal{CR}_{\mathcal{J}}(\mathcal{A}) \leqslant 1$, the problem of determining the competitive ratio reduces to finding $\upsilon = \sup\{\nu \in \mathbb{Q} : \ \Phi_\nu \text{ is satisfied in } G\}$. Because this value corresponds to the ratio of the corresponding rewards obtained in a simple cycle in G, it follows that υ is the maximum of a finite set, and can be determined exactly by an *adaptive binary search*.

4.3.2 Reduction for limit-average constraints

Finally, we turn our attention to additional limit-average constraints and the LTS $L_{\mathcal{W}}$. We follow a similar approach as above, but this time including $L_{\mathcal{W}}$ in the synchronous product, i.e., $L = L_{\mathcal{A}} \times L_{\mathcal{C}} \times L_{\mathcal{S}} \times L_{\mathcal{L}} \times L_{\mathcal{W}}$. Let $w_{\mathcal{A}}$ and $w_{\mathcal{C}}$ be weight functions that assign to each edge $e \in E$ in the corresponding multi-graph a vector of $d + 1$ weights as follows. In the first dimension, $w_{\mathcal{A}}$ and $w_{\mathcal{C}}$ are defined as before, assigning to each edge of G the corresponding rewards of $\mathcal{A}$ and $\mathcal{C}$. In the remaining d dimensions, $w_{\mathcal{C}}$ is always 1, whereas $w_{\mathcal{A}}$ equals the value of the weight function w of $L_{\mathcal{W}}$ on the corresponding transition. Let $\vec{\lambda}$ be the threshold vector of $L_{\mathcal{W}}$. It follows that for all $\nu \in \mathbb{Q}$, we have that $\mathcal{CR}_{\mathcal{J}}(\mathcal{A}) \leqslant \nu$ iff the objective $\Phi_\nu = \mathsf{Safe}(X) \cap \mathsf{Live}(Y) \cap \mathsf{Ratio}(w_{\mathcal{A}}, w_{\mathcal{C}}, (\nu, \vec{\lambda}))$ is satisfied in G from the state s that corresponds to the initial state of each LTS, where $(\nu, \vec{\lambda})$ is a $d+1$-dimension vector, with ν in the first dimension, followed by the d-dimension vector $\vec{\lambda}$. As the dimension in the ratio objective is greater than one, Case 2 of Theorem 1 applies, and we obtain the following:

Lemma 2 *Given the product graph $G = (V, E)$ of n nodes and m edges, a rational $\nu \in \mathbb{Q}$, and a set of job sequences $\mathcal{J}$ admissible for safety, liveness, and limit average LTSs, determining whether $\mathcal{CR}_{\mathcal{J}}(\mathcal{A}) \leqslant \nu$ requires polynomial time.*

Again, since $0 \leqslant \mathcal{CR}_{\mathcal{J}}(\mathcal{A}) \leqslant 1$, the competitive ratio is determined by an adaptive binary search. However, this time $\mathcal{CR}_{\mathcal{J}}(\mathcal{A})$ is not guaranteed to be realized by a simple cycle (the witness path in G is not necessarily periodic, see Algorithm 1), and is only approximated within some desired error threshold $\varepsilon > 0$.

4.3.3 Adaptive binary search

We employ an *adaptive* binary search for the competitive ratio in the interval [0, 1], which works as follows: the algorithm maintains an interval $[\ell, r]$ such that $\ell \leqslant \mathcal{CR}_{\mathcal{J}}(\mathcal{A}) \leqslant r$ at all times, and exploits the nature of the problem for refining the interval according to the following rules: first, if the current objective $\nu \in [\ell, r]$ (typically, $\nu = (\ell + r)/2$) is satisfied in G, i.e., Lemma 1 answers "yes" and provides the current minimum cycle C as a witness, the value r is updated to the ratio ν' of the on-line and off-line rewards in C, which is typically less than ν. This allows to reduce the current interval for the next iteration from $[\ell, r]$ to $[\ell, \nu']$, with $\nu' \leqslant \nu$, rather than $[\ell, \nu]$ (as a simple binary search would do). Second, since $\mathcal{CR}_{\mathcal{J}}(\mathcal{A})$ corresponds to the ratio of rewards on a simple cycle in G, if the current objective $\nu \in [\ell, r]$ is not satisfied in G, the algorithm assumes that $\mathcal{CR}_{\mathcal{J}}(\mathcal{A}) = r$ (i.e, the competitive ratio equals the right endpoint of the current interval), and tries $\nu = r$ in the next iteration. Hence, as opposed to a naive binary search, the adaptive version has the advantages of (i) returning the exact value of $\mathcal{CR}_{\mathcal{J}}(\mathcal{A})$ (rather than an approximation), and (ii) being faster in practice.

Remark 4 Lemmas 1 and 2 give polynomial upper bounds for the complexity of determining the competitive ratio of an online scheduling algorithm $\mathcal{A}$ given as a LTS $L_{\mathcal{A}}$. If, instead, $\mathcal{A}$ is given in some succinct form using a description which is polylogarithmic in the number of states [e.g., as a circuit (Galperin and Wigderson 1983)], then the corresponding upper bounds become exponential in the size of the description of $\mathcal{A}$.

4.4 Optimized reduction

In Sect. 4.3, we established a formal reduction from determining the competitive ratio of an on-line scheduling algorithm in a constrained adversarial environment to solving multiple objectives on graphs. In this section, we present several optimizations for this reduction that significantly reduce the size of the generated LTSs.

4.4.1 Clairvoyant LTS reduction

Recall the clairvoyant LTS $L_{\mathcal{C}}$ with reward function $r_{\mathcal{C}}$ from Sect. 3, which non-deterministically models a scheduler. For our optimization, we encode the off-line algorithm as a non-deterministic LTS $L'_{\mathcal{C}} = (S'_{\mathcal{C}}, s'_{\mathcal{C}}, \Sigma, \varnothing, \Delta'_{\mathcal{C}})$ with reward function $r'_{\mathcal{C}}$ that lacks the property of being a scheduler, as information about released and scheduled jobs is lost. However, it preserves the property that, given a job sequence σ, there exists a run $\rho^{\sigma}_{\mathcal{C}}$ in $L_{\mathcal{C}}$ iff there exists a run $\widehat{\rho}^{\sigma}_{\mathcal{C}}$ in $L'_{\mathcal{C}}$ with $V(\rho^{\sigma}_{\mathcal{A}}, k) = V(\widehat{\rho}^{\sigma}_{\mathcal{A}}, k)$ for all $k \in \mathbb{N}^+$. That is, there is a bisimulation between $L_{\mathcal{C}}$ and $L'_{\mathcal{C}}$ that preserves rewards.

Intuitively, the clairvoyant algorithm need not partially schedule a job, i.e., it will either discard it immediately, or schedule it to completion. Hence, in every release of a set of tasks T, $L'_{\mathcal{C}}$ non-deterministically chooses a subset $T' \subseteq T$ to be scheduled, as well as allocates the future slots for their execution. Once these slots are allocated,

$L'_{\mathcal{C}}$ is not allowed to preempt those in favor of a subsequent job. For the reward, we use $r'_{\mathcal{C}} = \sum_{\tau_i \in T'} V_i$.

The state space $S'_{\mathcal{C}}$ of $L'_{\mathcal{C}}$ consists of binary strings of length $D_{\max}$. For a binary string $B \in S'_{\mathcal{C}}$, we have $B[i] = 1$ iff the i-th slot in the future is allocated to some released job, and $s'_{\mathcal{C}} = \vec{\mathbf{0}}$. Informally, the transition relation $\Delta'_{\mathcal{C}}$ is such that, given a current subset $T \subseteq \Sigma$ of newly released jobs, there exists a transition δ from B to B' only if B' can be obtained from B by non-deterministically choosing a subset $T' \subseteq T$, and for each task $\tau_i \in T'$ allocating non-deterministically C_i free slots in B.

By definition, $|S'_{\mathcal{C}}| \leqslant 2^{D_{\max}}$. In laxity-restricted tasksets, however, we can obtain an even tighter bound: let $L_{\max} = \max_{\tau_i \in \mathcal{T}} (D_i - C_i)$ be the maximum laxity in $\mathcal{T}$, and $I : S'_{\mathcal{C}} \to \{\perp, 1, \ldots, D_{\max} - 1\}^{L_{\max}+1}$ denote a function such that $I(B) = (i_1, \ldots, i_{L_{\max}+1})$ are the indexes of the first $L_{\max} + 1$ zeros in B. That is, $i_j = k$ iff $B[k]$ is the j-th zero location in B, and $i_j = \perp$ if there are less than j free slots in B.

Claim The function I is bijective.

Proof Fix a tuple $(i_1, \ldots, i_{L_{\max}+1})$ with $i_j \in \{\perp, 1, \ldots, D_{\max} - 1\}$, and let $B \in S'_{\mathcal{C}}$ be any state such that $I(B) = (i_1, \ldots, i_{L_{\max}+1})$. We consider two cases.

1. If $i_{L_{\max}+1} = \perp$, there are less than $L_{\max} + 1$ empty slots in B, all uniquely determined by $(i_1, \ldots, i_k)$, for some $k \leqslant L_{\max}$.
2. If $i_{L_{\max}+1} \neq \perp$, then all $i_j \neq \perp$, and thus any job to the right of $i_{L_{\max}+1}$ would have been stalled for more than $L_{\max}$ positions. Hence, all slots to the right of $i_{L_{\max}+1}$ are free in B, and B is also unique.

Hence, $I(B)$ always uniquely determines B, as desired. □

For $x, k \in \mathbb{N}^+$, denote with $\mathsf{Perm}(x, k) = x \cdot (x-1) \ldots (x-k+1)$ the number of k-permutations on a set of size x. Claim 4.4.1 immediately implies the following Lemma 3:

Lemma 3 *Let $\mathcal{T}$ be a taskset with maximum deadline $D_{\max}$, and $L_{\max} = \max_{\tau_i \in \mathcal{T}} (D_i - C_i)$ be the maximum laxity. Then, $|S'_{\mathcal{C}}| \leqslant \min(2^{D_{\max}}, \mathsf{Perm}(D_{\max}, L_{\max} + 1))$.*

Hence, for zero and small laxity environments (Baruah et al. 1992), as they typically arise in high-speed network switches (Englert and Westermann 2007) and in NoCs (Lu and Jantsch 2007), $S'_{\mathcal{C}}$ has polynomial size in $D_{\max}$. This affects the parameter n in Lemmas 1 and 2.

4.4.2 Clairvoyant LTS generation

We now turn our attention to efficiently generating the clairvoyant LTS $L'_{\mathcal{C}}$ as described in the previous paragraph. There is non-determinism in two steps: both in choosing the subset $T' \subseteq T$ of the currently released tasks for execution, and in allocating slots for executing all tasks in T'. Given a current state B and T, this non-determinism leads to several identical transitions δ to a state B'. We have developed a recursive algorithm called ClairvoyantSuccessor (Algorithm 2) that generates each such transition δ exactly once.

Algorithm 2: ClairvoyantSuccessor

Input: A set $T \subseteq \mathcal{T}$, state B, index $1 \leqslant k \leqslant D_{\max}$
Output: A set $\mathcal{B}$ of successor states of B

```
1  if T = ∅ then return {B};
2  τ ← arg min_{τ_i ∈ T} D_i, C ← execution time of τ
3  T' ← T \ {τ}
   // Case 1: τ is not scheduled
4  B ← ClairvoyantSuccessor(T', B, k)
   // Case 2: τ is scheduled
5  F ← set of free slots in B greater than k
6  foreach F ⊆ F with |F| = C do
7      B' ← Allocate F in B
8      k' ← rightmost slot in F
9      B' ← ClairvoyantSuccessor(T', B', k')
       // Keep only non-redundant states
10     foreach B'' ∈ B' do
11         if B''[1] = 1 and knapsack(B'', T) then
12             B ← B ∪ {B''}
13         end
14     end
15 end
16 return B
```

The intuition behind ClairvoyantSuccessor is as follows: it is well-known that the earliest deadline first (EDF) policy is optimal for scheduling job sequences where every released task can be completed (Dertouzos 1974). By construction, given a job sequence σ_1, $L'_{\mathcal{C}}$ non-deterministically chooses a job sequence σ_2, such that for all ℓ, we have $\sigma_2^{\ell} \subseteq \sigma_1^{\ell}$, and all jobs in σ_2 are scheduled to completion by $L'_{\mathcal{C}}$. Therefore, it suffices to consider a transition relation $\Delta'_{\mathcal{C}}$ that allows at least all possible choices that admit a feasible EDF schedule on every possible σ_2, for any generated job sequence σ_1.

In more detail, ClairvoyantSuccessor is called with a current state B, a subset of released tasks T and an index k, and returns the set $\mathcal{B}$ of all possible successors of B that schedule a subset $T' \subseteq T$, where every job of T' is executed later than k slots in the future. This is done by extracting from T the task τ with the earliest deadline, and proceeding as follows: the set $\mathcal{B}$ is obtained by constructing a state B' that considers all the possible ways to schedule τ to the right of k (including the possibility of not scheduling τ at all), and recursively finding all the ways to schedule $T \setminus \{\tau\}$ in B', to the right of the rightmost slot allocated for task τ.

Finally, we exploit the following two observations to further reduce the state space of $L'_{\mathcal{C}}$. First, we note that as long as there are some unfinished jobs in the state of $L'_{\mathcal{C}}$ (i.e., at least one bit of B is one), the clairvoyant algorithm gains no benefit by not executing any job in the current slot. Hence, besides the zero state $\vec{\mathbf{0}}$, every state B must have $B[1] = 1$. In most cases, this restriction reduces the state space by at least 50%.

Second, observe that for every two scheduled jobs J and J', the clairvoyant scheduler will never have to preempt J for J' and vice versa. Given a state B, we call a

contiguous segment of zeros in B which is surrounded by ones a *gap*. We call a gap between positions $[i_1, i_2]$ of B *admissible* if there exists a multiset X of tasks from $\mathcal{T}$ such that $\sum_{\tau_i \in X} C_i = i_2 - i_1 + 1$. Observe that if state B contains a gap which is not admissible, then the clairvoyant scheduler produces a schedule in which either

1. no job is scheduled in some round, while there is some already released job J which will be scheduled in the future, or
2. two jobs J and J' are such that each one preempts the other.

It is straightforward that in both cases, the clairvoyant scheduler can obtain the same utility by producing another schedule in which none of the above cases occur. Hence, a state can be safely discarded if it contains a non-admissible gap. This reduces to solving a knapsack problem (Karp 1972), where the size of the knapsack is the length of the gap, and the set of items is the whole taskset $\mathcal{T}$ (with multiplicities). We note that the problem has to be solved on identical inputs a large number of times, and techniques such as *memoization* are employed to avoid multiple evaluations of the same input.

These two improvements were found to reduce the state space by a factor up to 90% in all examined cases (see Sect. 4.5 and Table 5). In fact, despite the non-determinism, in all reported cases the generation of $L_{\mathcal{C}}$ was done in less than a second.

4.4.3 On-line LTS reduction

Typically, simple on-line scheduling algorithms do "lazy dropping" of unsuccessful jobs, where such a job is dropped only when its deadline passes. An obvious improvement for reducing the size of the state space of the LTS is to implement some early dropping: store only those jobs that could be scheduled, at least partially, under *some* sequence of future task releases. We do so by first creating the LTS naively, and then iterating through its states. For each state s and job $J_{i,j}$ in s with relative deadline D_i, we perform a *depth-limited search* originating in s for D_i steps, looking for a state s' reached by a transition that schedules $J_{i,j}$. If no such state is found, we merge state s to s'', where s'' is identical to s without job $J_{i,j}$.

4.5 Experimental results

We have implemented a prototype of our automated competitive ratio analysis framework, and applied it in a comparative case study.

Our implementation has been done in Python 2.7 and C, and uses the lp_solve (Berkelaar et al. 2004) package for linear programming solutions. All experiments were run on a standard desktop computer with a 3.2 GHz CPU and 4 GB of RAM running Debian Linux.

In our case study, five well-known scheduling policies, namely, EDF (earliest deadline first), LLF (least laxity first), SRT (shortest remaining time), SP (static priorities), and FIFO (first-in first-out), as well as some more elaborate algorithms that provide non-trivial performance guarantees, in particular, DSTAR (Baruah et al. 1991), TD1 (Baruah et al. 1992), and DOVER (Koren and Shasha 1995), were analyzed

under a variety of tasksets (with and without additional constraints on the adversary). In addition, for TD1, we constructed a series of task sets according to the recurrence relation given in Baruah et al. (1992), which confirms its worst-case competitive ratio of 1/4. All our on-line scheduler implementations use the same tie-breaking rules, namely, (i) favor lower-indexed tasks (in $\mathcal{T}$) over higher-indexed ones, and (ii) favor smaller deadlines over larger ones [and (i) has higher precedence over (ii)].

Varying tasksets without constraints The algorithm DOVER was proved in Koren and Shasha (1995) to have optimal competitive factor, i.e., optimal competitive ratio under the worst-case taskset. However, our experiments reveal that this performance guarantee is not universal, in the sense that DOVER is outperformed by other schedulers for *specific* tasksets. Interestingly, this observation applies to all on-line algorithms examined: as shown in Fig. 6, even without constraints on the adversary, there are tasksets in which every chosen scheduling algorithm outperforms all others, by achieving the highest competitive ratio for the particular taskset. This sensitivity of the optimally performing on-line algorithm on the given taskset makes our automated analysis framework a very interesting tool for the application designer.

Table 1 lists the tasksets A1–A7 used for Fig. 6. The task indices, hence their order in Table 1, reflect their static priorities (with τ_1 having highest priority); they are used by the SP scheduler, as well as for tie breaking by other schedulers. Along with each tasket, its importance ratio $k = \frac{\max_{\tau_i \in \mathcal{T}}\{V_i/C_i\}}{\min_{\tau_i \in \mathcal{T}}\{V_i/C_i\}}$ is shown (Baruah et al. 1992).

Fixed taskset with varying constraints We also analyzed fixed tasksets under various constraints (such as sporadicity or workload restrictions) for admissible job sequences. Figure 7 shows some experimental results for workload safety constraints, which again reveal that, depending on particular workload constraints, we can have different

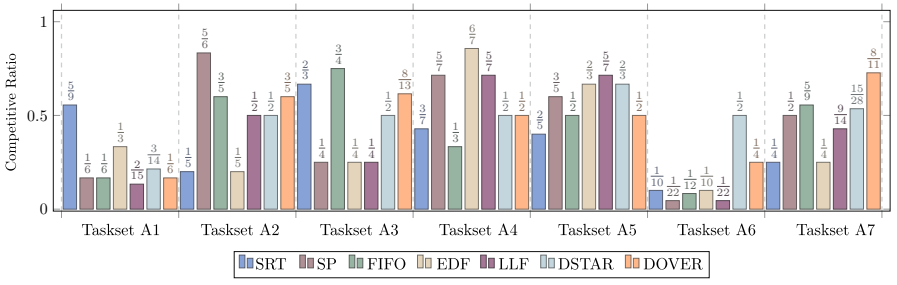


Fig. 6 The competitive ratio of the examined algorithms in various tasksets under no constraints. Every examined algorithm is optimal in some taskset, among all others

Table 1 The tasksets used to generate Fig. 6

	A1 ($k = 6$)				A2 ($k = 5$)		A3 ($k = 4$)			A4 ($k = 3$)			A5 ($k = 2$)			A6 ($k = 4$)			A7 ($k = 5$)		
	τ_1	τ_2	τ_3	τ_4	τ_1	τ_2	τ_1	τ_2	τ_3	τ_1	τ_2	τ_3	τ_1	τ_2	τ_3	τ_1	τ_2	τ_3	τ_1	τ_2	τ_3
C_i	1	4	1	3	2	2	2	1	1	1	2	1	2	1	1	2	6	1	1	2	1
D_i	2	6	3	4	3	2	2	5	5	2	3	6	3	1	3	2	6	1	5	2	1
V_i	3	2	3	3	5	1	1	2	2	3	2	1	9	6	3	1	10	2	5	4	1

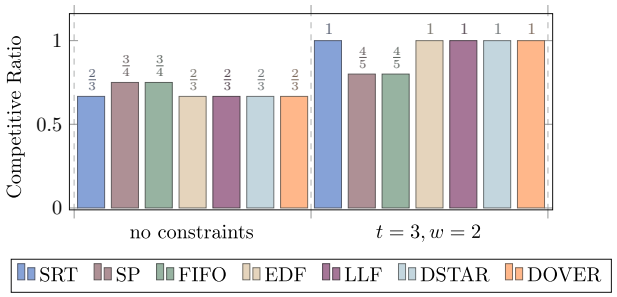


Fig. 7 Restricting the absolute workload generated by the adversary typically increases the competitive ratio, and can vary the optimal scheduler. On the left, the performance of each scheduler is evaluated without restrictions: FIFO, SP behave best. When restricting the adversary to at most 2 units of workload in the last 3 rounds, FIFO and SP become suboptimal, and are outperformed by other schedulers

Table 2 Columns show the mean workload restriction. The check-marks indicate that the corresponding scheduler is optimal for that mean workload restriction, among the six schedulers we examined. We see that the optimal scheduler can vary as the restrictions are tighter, and in a non-monotonic way. LLF, EDF, DSTAR and DOVER were not optimal in any case and hence not mentioned

	1.5	1	0.8	0.6	0.4	0.3	0.1	0.078	0.05
FIFO	✓	✓	✓	✓	✓				✓
SP	✓						✓		✓
SRT	✓				✓	✓	✓	✓	✓

Table 3 Taskset of Fig. 7 (left) and Table 2 (right)

	τ_1	τ_2	τ_3
C_i	1	1	1
D_i	1	2	1
V_i	3	3	1
C_i	2	5	5
D_i	7	5	6
V_i	3	2	1

optimal schedulers. The same was observed for limit-average constraints: as Table 2 shows, the optimal scheduler can vary highly and non-monotonically with stronger limit-average workload restrictions. The tasksets for both experiments are shown in Table 3.

Competitive ratio of TD1 We also analyzed the performance of the on-line scheduler TD1 for zero laxity tasksets with uniform value-density $k = 1$ (i.e., $C_i = D_i = V_i$ for each task τ_i). Following (Baruah et al. 1992), we constructed a series of tasksets parametrized by some positive real $\eta < 4$, which guarantee that the competitive ratio of every on-line scheduler is upper bounded by $\frac{1}{\eta}$. Given η, each taskset consists of tasks τ_i such that C_i is given by the following recurrence, as long as $C_{i+1} > C_i$.

Table 4 Competitive ratio of TD1

Taskset	η	Taskset	Comp. Ratio
C1	2	{1, 1}	1
C2	3	{1, 2, 3}	1/2
C3	3.1	{1, 3, 7, 13, 19}	7/25
C4	3.2	{1, 3, 7, 13, 20, 23}	1/4
C5	3.3	{1, 3, 7, 14, 24, 33}	1/4
C6	3.4	{1, 3, 7, 14, 24, 34}	1/4

Table 5 Scalability of our approach for tasksets of various sizes N and $D_{\max}$. For each taskset, the size of the state space of the clairvoyant scheduler is shown, along with the mean size of the product LTS, and the mean and maximum time to solve one instance of the corresponding ratio objective

Taskset	N	$D_{\max}$	Size (nodes)		Time (s)	
			Clairv.	*Product*	*Mean*	*Max*
B01	2	7	19	823	0.04	0.05
B02	2	8	26	1997	0.39	0.58
B03	2	9	34	4918	10.02	15.21
B04	3	7	19	1064	0.14	0.40
B05	3	8	26	1653	0.66	2.05
B06	3	9	34	7705	51.04	136.62
B07	4	7	19	1711	2.13	6.34
B08	4	8	26	3707	13.88	34.12
B09	4	9	44	10, 040	131.83	311.94
B10	5	7	19	2195	5.73	16.42
B11	5	8	32	9105	142.55	364.92
B12	5	9	44	16, 817	558.04	1342.59

$$(i)\ C_0 = 1 \qquad (ii)\ C_{i+1} = \eta \cdot C_i - \sum_{j=0}^{i} C_j$$

In Baruah et al. (1992), TD1 was shown to have competitive factor $\frac{1}{4}$, and hence a competitive ratio that approaches $\frac{1}{4}$ from above, as $\eta \to 4$ in the above series of tasksets. Table 4 shows the competitive ratio of TD1 in our constructed series of tasksets. Each taskset is represented as a set $\{C_i : 1 \leqslant i \leqslant n\}$, where each C_i is given by the above recurrence, rounded up to the next integer. We indeed see that the competitive ratio drops until it stabilizes to $\frac{1}{4}$. Note that, thanks to our optimizations, the zero-laxity restriction allowed us to process tasksets where $D_{\max}$ is much higher than for the tasksets reported in Table 5: the results of Table 4 were produced in less than a minute overall.

Running times Table 5 summarizes some key parameters of our various tasksets, and gives some statistical data on the observed running times in our respective experiments. Even though the considered tasksets are small, the very short running times of our prototype implementation reveal the principal feasibility of our approach. We believe that further application-specific optimizations, augmented by abstraction and symmetry reduction techniques, will allow to scale to larger applications.

5 Competitive synthesis of on-line scheduling algorithms

In this section, we show how the powerful framework of *graph games* (Martin 1975; Shapley 1953) can be utilized for the *synthesis* of optimal real-time scheduling algorithms. As opposed to the the analysis problem considered in the previous sections (which can be viewed as a 1-player game of the adversary against a given scheduling algorithm), we now have to consider a two-player game between the (sought) optimal on-line algorithm (Player 1) and the adversary (Player 2). Our presentation is organized as follows:

- In Sect. 5.1, we introduce a suitable two-player partial-information game with mean-payoff and ratio objectives. Player 1 will represent the online algorithm, whereas Player 2 will represent both the adversary (which chooses the job sequence) and the clairvoyant algorithm (which knows the job sequence in advance). We use a partial-information setting to model that Player 1 is oblivious to the scheduling choices of Player 2, but Player 2 knows the scheduling choices of Player 1 for deciding which future jobs to release. The mean-payoff and ratio objectives model directly the worst-case utility and competitive ratio problems, respectively.
- In Sect. 5.2, we establish that the relevant decision problems for our game are NP-complete in the size of the game graph.
- In Sect. 5.3, we study the decision problems relevant for two particular synthesis questions: in *synthesis for worst-case average utility*, the goal is to automatically construct an on-line scheduling algorithm with the largest possible worst-case average utility for a given taskset. In *competitive synthesis*, we construct an on-line scheduling algorithm with the largest possible competitive ratio for the given taskset. The complexity results for our graph game reveal that the former problem is in NP∩coNP, whereas the latter is in NP. These complexities are wrt the size of the constructed algorithm, represented explicitly as a labeled transition system. As a function of the input taskset $\mathcal{T}$ given in binary, all polynomial upper bounds become exponential upper bounds in the worst case. The solution to the competitive synthesis . Hence the algorithm for obtaining the optimal scheduler comes in two steps. The first step reduces the problem to the relevant partial-information game and is found in Theorem 5 of Sect. 5.3. The second step is solving the partial-information game, and is found in Theorem 3 of Sect. 5.2.

5.1 Partial-information mean-payoff and ratio games

We first introduce a two-player partial-information game on graphs with mean-payoff and ratio objectives.

5.1.1 Notation on graph games

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

State space The set S is a finite set of states.

Actions	Σ_i ($i = 1, 2$) is a finite set of actions for Player i.
Transition function	Given the current state $s \in S$, an action $\alpha_1 \in \Sigma_1$ for Player 1, and an action $\alpha_2 \in \Sigma_2$ for Player 2, the transition function $\delta : S \times \Sigma_1 \times \Sigma_2 \to S$ gives the next (or successor) state $s' = \delta(s, \alpha_1, \alpha_2)$. A shorter form to denote a transition is to write the tuple $(s, \alpha_2, \alpha_1, s')$; note that α_2 is listed before α_1 to stress that fact that Player 2 chooses its action before Player 1.
Observations	The set $\mathcal{O}_S \subseteq 2^S$ is a finite set of observations for Player 1 that partition the state space S. This partition uniquely defines a function $\mathsf{obs}_S : S \to \mathcal{O}_S$, which maps each state to its observation $\mathsf{obs}_S(s)$ in a way that ensures $s \in \mathsf{obs}_S(s)$ for all $s \in S$. 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, and analogously defines the function obs_Σ. Intuitively, Player 1 will have partial observation, and can only obtain 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.

5.1.2 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 according to the transition function δ.

A *play* in $\mathcal{G}$ is an infinite sequence of states and actions $\mathscr{P} = s^1, \alpha_2^1, \alpha_1^1, s^2, \alpha_2^2, \alpha_1^2, s^3, \alpha_2^3, \alpha_1^3, s^4 \ldots$ such that, for all $j \geqslant 1$, we have $\delta(s^j, \alpha_1^j, \alpha_2^j) = s^{j+1}$. The *prefix up to* s^n of the play $\mathscr{P}$ is denoted by $\mathscr{P}(n)$ and corresponds to the starting state of the n-th turn. The set of plays in $\mathcal{G}$ is denoted by $\mathscr{P}^\infty$, and the set of corresponding finite prefixes is denoted by $\mathsf{Prefs}(\mathscr{P}^\infty)$.

5.1.3 Strategies

A *strategy* for a player is a recipe that specifies how to extend finite prefixes of plays. We will consider *memoryless* deterministic strategies for Player 1 (where its next action depends only on the current state, but not on the entire history) and general history-dependent deterministic strategies for Player 2. A strategy for Player 1 is a function $\pi : \mathcal{O}_S \times \mathcal{O}_\Sigma \to \Sigma_1$ 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 $\sigma : \mathsf{Prefs}(\mathscr{P}^\infty) \to \Sigma_2$ 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 current observations (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 $\Pi_{\mathcal{G}}^M$, $\Sigma_{\mathcal{G}}$, $\Sigma_{\mathcal{G}}^M$ 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 strategy σ for Player 1 and Player 2, and an initial state s^1, we obtain a unique play $\mathscr{P}(s^1, \pi, \sigma) = s^1, \alpha_2^1, \alpha_1^1, s^2, \alpha_2^2, \alpha_1^2, s^3, \ldots$ such that, for all $n \geqslant 1$, we have $\sigma(\mathscr{P}(n)) = \alpha_2^n$ and $\pi(\mathsf{obs}_S(s^n), \mathsf{obs}_\Sigma(\alpha_2^n)) = \alpha_1^n$.

5.1.4 Objectives

Recall that, for the graphs with multiple objectives from Sect. 4.1, an objective is a set of paths. Here we extend this notion to games: an objective of a game $\mathscr{G}$ is a set of plays that satisfy some desired properties. For the sake of completeness, we present here the relevant definitions for mean payoff and ratio objectives with 1-dimensional weight functions.

For mean-payoff objectives, we will consider a reward function $w : S \times \Sigma_1 \times \Sigma_2 \times S \to \mathbb{Z}$ that maps every transition to an integer reward. The reward function naturally extends to plays: for $k \geqslant 1$, the sum of the rewards in the prefix $\mathscr{P}(k+1)$ is defined as $w(\mathscr{P}, k) = \sum_{i=1}^{k} w(s^i, \alpha_2^i, \alpha_1^i, s'^i)$. The mean-payoff of a play $\mathscr{P}$ is then

$$\mathsf{MP}(w, \mathscr{P}) = \liminf_{k \to \infty} \frac{1}{k} \cdot w(\mathscr{P}, k).$$

In the case of ratio objectives, we will consider two reward functions $w_1 : S \times \Sigma_1 \times \Sigma_2 \times S \to \mathbb{N}$ and $w_2 : S \times \Sigma_1 \times \Sigma_2 \times S \to \mathbb{N}$ that map every transition to a non-negative valued reward. Using the same extension of reward functions to plays as before, the ratio of a play $\mathscr{P}$ is defined as:

$$\mathsf{Ratio}(w_1, w_2, \mathscr{P}) = \liminf_{k \to \infty} \frac{\vec{\mathbf{1}} + w_1(\mathscr{P}, k)}{\vec{\mathbf{1}} + w_2(\mathscr{P}, k)}.$$

5.1.5 Decision problems

Analogous to Sect. 4.1, we define the relevant decision problems on games. Formally, given a game $\mathscr{G}$, a starting state s^1, reward functions w, w_1, w_2 and a threshold $\nu \in \mathbb{N}$, the decision problem for the mean payoff objective is to decide whether

$$\sup_{\pi \in \Pi_{\mathscr{G}}^M} \inf_{\sigma \in \Sigma_{\mathscr{G}}} \mathsf{MP}(w, \mathscr{P}(s^1, \pi, \sigma)) \geqslant \nu.$$

Similarly, the decision problem for the ratio objective is to decide whether

$$\sup_{\pi \in \Pi_{\mathscr{G}}^M} \inf_{\sigma \in \Sigma_{\mathscr{G}}} \mathsf{Ratio}(w_1, w_2, \mathscr{P}(s^1, \pi, \sigma)) \geqslant \nu.$$

Remark 5 Note that the decision problems of the graph game problem are defined over the $\sup_{\pi \in \Pi_{\mathscr{G}}^M}$, taking all possible memoryless strategies into account. This corresponds to all possible on-line scheduling strategies, whereas the multi-graph problem arising

in the competitive analysis problem considered in the previous sections explicitly used the fixed deterministic strategy for the on-line scheduler only.

5.1.6 Perfect-information games

Games of complete-observation (or perfect-information games) are a special case of partial-observation games where $\mathcal{O}_S = \{\{s\} \mid s \in S\}$ and $\mathcal{O}_\Sigma = \{\{\alpha_2\} \mid \alpha_2 \in \Sigma_2\}$, i.e., every individual state and action is fully 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. The following theorem for perfect-information games with mean-payoff objectives follows from the results of Ehrenfeucht and Mycielski (1979), Zwick and Paterson (1996), Brim et al. (2011), Karp (1978).

Theorem 2 (Complexity of perfect-information mean-payoff games) (Ehrenfeucht and Mycielski 1979; Zwick and Paterson 1996; Brim et al. 2011; Karp 1978). *The following assertions hold for perfect-information games with initial state s^1 and reward function $w : S \times \Sigma_1 \times \Sigma_2 \times S \to \mathbb{Z}$:*

1. (*Determinacy*) We have

$$\begin{aligned} &\sup_{\pi \in \Pi_{\mathcal{G}}^M} \inf_{\sigma \in \Sigma_{\mathcal{G}}} \mathsf{MP}(w, \mathscr{P}(s^1, \pi, \sigma)) \\ = &\inf_{\sigma \in \Sigma_{\mathcal{G}}} \sup_{\pi \in \Pi_{\mathcal{G}}^M} \mathsf{MP}(, w, \mathscr{P}(s^1, \pi, \sigma)) \\ = &\inf_{\sigma \in \Sigma_{\mathcal{G}}^M} \sup_{\pi \in \Pi_{\mathcal{G}}^M} \mathsf{MP}(w, \mathscr{P}(s^1, \pi, \sigma)). \end{aligned}$$

2. Whether $\sup_{\pi \in \Pi_{\mathcal{G}}^M} \inf_{\sigma \in \Sigma_{\mathcal{G}}} \mathsf{MP}(w, \mathscr{P}(s^1, \pi, \sigma)) \geqslant \nu$ can be decided in NP $\cap$ coNP, for a rational threshold ν.
3. The computation of the optimal value $v^* = \sup_{\pi \in \Pi_{\mathcal{G}}^M} \inf_{\sigma \in \Sigma_{\mathcal{G}}} \mathsf{MP}(w, \mathscr{P}(s^1, \pi, \sigma))$ and an optimal memoryless strategy $\pi^* \in \Pi_{\mathcal{G}}^M$ such that $v^* = \inf_{\sigma \in \Sigma_{\mathcal{G}}} \mathsf{MP}(w, \mathscr{P}(s^1, \pi^*, \sigma))$ 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 maximum value of all the rewards (i.e., the algorithm runs in pseudo-polynomial time, and if the maximum value W of rewards is polynomial in the size of the game, then the algorithm is polynomial).

5.1.7 Sketch of the algorithm

The complexity of Item 3 of Theorem 2 is obtained in Brim et al. (2011). Here we outline a simple algorithm for solving the same problem in time $O(n^4 \cdot m \cdot \log(n/m) \cdot W)$, as found in Zwick and Paterson (1996). The algorithm operates in two steps. First, we compute for every node $u \in S$ the maximum mean payoff $v(u)$ that Player 1 can ensure in any play that starts from u. This is achieved by the standard value-iteration procedure executed for $\Theta(n^2 \cdot W)$ iterations. Hence, the time required for this step is $O(n^2 \cdot m \cdot W)$. Note that at this point $v(s^1)$ gives the mean payoff achieved by an

optimal strategy $\pi^* \in \Pi^M_{\mathscr{G}}$, but not the strategy itself. Since an optimal *memoryless* strategy is guaranteed to exist, this strategy can be computed by a binary search on the actions of Player 1. Given a node $u \in S$, we denote by $\Sigma_1(u)$ the set of actions available to Player 1 on u. In the second step, we iteratively pick a node $u \in S$ which has more than one available actions for Player 1, and a set $X \subset \Sigma_1(u)$ which contains half of the actions of Player 1 on u. We let $\mathscr{G}'$ be the modified game where the actions for Player 1 on node u is the set X, and recompute the value $v'(u)$ in $\mathscr{G}'$. If $v'(u) = v(u)$, we repeat the process on $\mathscr{G}'$. Otherwise, we construct a new game $\mathscr{G}''$ which is identical to $\mathscr{G}$, but such that the available actions for Player 1 on node u is the set $\Sigma_1(u) \setminus X$. We repeat the process on $\mathscr{G}''$.

5.2 Complexity results

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

5.2.1 Transformation

We start with a simple transformation that will allow us to technically simplify our proof. In our definition of games, every action was available for the players in every state for simplicity. We will now consider restricted games where, in certain states, some actions are not allowed for a player. The transformation of such restricted games to games where all actions are allowed is as follows: we add two absorbing dummy states (with only a self-loop), one for Player 1 and the other for Player 2, and assign rewards in a way such that the objectives are violated for the respective player. For example, for mean-payoff objectives with threshold $\nu > 0$, we assign reward 0 for the only out-going (self-loop) transition of the Player 1 dummy state, and a reward strictly greater than ν for the self-loop of the Player 2 dummy state; in the 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 go to the dummy Player 1 state; and if Player 2 plays an action that is not allowed, we go to the Player 2 dummy state. Obviously, this is a simple linear time transformation. Hence, for technical convenience, we can assume in the sequel that different states have different sets of available actions for the players. We first start with the hardness result.

Lemma 4 *The decision problems for partial-observation games with mean-payoff objectives and ratio objectives, i.e., whether* $\sup_{\pi \in \Pi^M_{\mathscr{G}}} \inf_{\sigma \in \Sigma_{\mathscr{G}}} \mathsf{MP}(w, \mathscr{P}(s^1, \pi, \sigma)) \geqslant \nu$ *(respectively* $\sup_{\pi \in \Pi^M_{\mathscr{G}}} \inf_{\sigma \in \Sigma_{\mathscr{G}}} \mathsf{Ratio}(w_1, w_2, \mathscr{P}(s^1, \pi, \sigma)) \geqslant \nu$*), 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 (Papadimitriou 1993). Let Ψ be a 3-SAT formula over n variables $x_1, x_2, \ldots, x_n$ in conjunctive normal form, with m clauses $\mathfrak{c}_1, \mathfrak{c}_2, \ldots, \mathfrak{c}_m$ consisting of a disjunction

of 3 literals (a variable x_k or its negation $\overline{x}_k$) each. We will construct a game graph $\mathcal{G}_\Psi$ as follows:

State space $S = \{s_{\mathsf{init}}\} \cup \{s_{i,j} \mid 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant 3\} \cup \{\mathsf{dead}\}$; i.e., there is an initial state s_{init}, a dead state dead, and there is a state $s_{i,j}$ for every clause $\mathfrak{c}_i$ and literal j of $\mathfrak{c}_i$.

Actions The set of actions applicable for Player 1 is $\{\mathsf{true}, \mathsf{false}, \bot\}$, the possible actions for Player 2 are $\{1, 2, \ldots, m\} \cup \{\bot\}$.

Transitions In the initial state s_{init}, Player 1 has only one action $\bot$ available, Player 2 has actions $\{1, 2, \ldots, m\}$ available, and given action $1 \leqslant i \leqslant m$, 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 $\mathfrak{c}_i$ is x_k, then we have a transition back to the initial state; and (ii) if the j-th literal in $\mathfrak{c}_i$ 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 $\mathfrak{c}_i$ 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 $\mathfrak{c}_i$ 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).

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 $\mathfrak{c}_i$ is either x_k or its negation $\overline{x}_k$, i.e., the observation for Player 1 corresponds to the variables.

A pictorial description is shown in Fig 8. 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. Note that Player 1 strategies are observation-based memoryless; hence, for every observation (i.e., a variable), it chooses a unique action (i.e., an assignment) and thus non-conflicting assignments are ensured. We consider $\mathcal{G}_\Psi$ with reward functions w, w_1, w_2 as follows: w_2 assigns reward 1 to all transitions; w and w_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 ratio objective). We now present the two directions of the proof.

Satisfiable implies dead is not reached: we show that if Ψ is satisfiable, then Player 1 has an observation-based memoryless strategy π^* to ensure that dead is

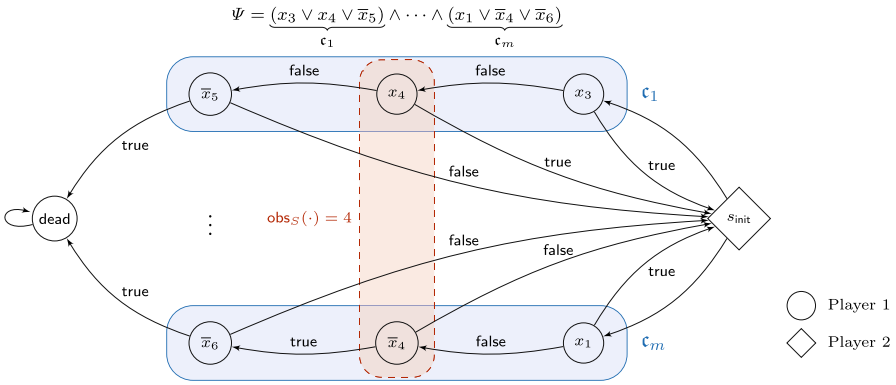


Fig. 8 Illustration of the construction of a game from a 3-SAT formula

never reached. Consider a satisfying assignment A for Ψ, 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 $1 \leqslant i \leqslant m$, 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 Ψ 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 it chooses false. Since π^* ensures that dead is not reached, it means for every $1 \leqslant i \leqslant m$, that there exists $s_{i,j}$ such that the transition to s_{init} is chosen (which ensures that $\mathfrak{c}_i$ is satisfied by A). Thus since π^* ensures dead is not reached, the assignment A is a satisfying assignment for Ψ.

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

5.2.2 The NP *upper bounds*

We now present the NP upper bounds for our decision problems. Recall that according to our definitions of strategies, 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). Such a strategy π can be guessed in polynomial time. Once the memoryless strategy is guessed and fixed, we need to show that there is a polynomial-time verification procedure:

Mean-payoff objectives Once the memoryless strategy for Player 1 is fixed, the game problem reduces to a 1-player game where there is only Player 2. The verification problem hence reduces to the path problem in directed graphs analyzed and shown to be solvable in polynomial time by Theorem 1 in Sect. 4.1.

Ratio objectives Again, once the memoryless strategy for Player 1 is fixed, the game problem reduces to a decision problem on directed graphs. The same reduction from ratio objectives to mean-payoff objectives introduced in Sect. 4.1 can be applied. Theorem 1 hence gives a polynomial-time verification algorithm for our ratio objectives.

We summarize the result in the following theorem.

Theorem 3 *The decision problems for partial-observation games with mean-payoff objectives and ratio objectives, i.e., whether* $\sup_{\pi \in \Pi^M_{\mathcal{G}}} \inf_{\sigma \in \Sigma_{\mathcal{G}}} \mathsf{MP}(w, \mathscr{P}(s^1, \pi, \sigma)) \geqslant \nu$ *respectively* $\sup_{\pi \in \Pi^M_{\mathcal{G}}} \inf_{\sigma \in \Sigma_{\mathcal{G}}} \mathsf{Ratio}(w_1, w_2, \mathscr{P}(s^1, \pi, \sigma)) \geqslant \nu$, *are* NP*-complete.*

Remark 6 The NP-completeness of Theorem 3 also holds with the following extensions on objectives:

1. The reward functions w, w_1, w_2 map to d-dimensional vectors of rewards, and the decision problems are with respect to a threshold vector $\vec{\nu}$.
2. Player 2 must also satisfy a conjunction of $\mathsf{Safe}(X)$ and $\mathsf{Live}(Y)$ objectives (see Sect. 4.1).

The result holds, as the NP-hardness follows from the proof of Theorem 3 by taking $d = 1$, $X = \varnothing$ and $Y = S$. The NP-membership follows similarly to that used in the proof of Theorem 3, by guessing a memoryless strategy for Player 1. The problem reduces to satisfying a conjunction of objectives in a multi-graph here, and Item 2 of Theorem 1 provides the required polynomial time bound.

5.3 Reduction of competitive synthesis to a graph game

We now turn our attention to competitive synthesis problems in the real-time scheduling context. More specifically, given a taskset $\mathcal{T}$, we consider two particular synthesis questions:

1. In *synthesis for the worst-case average utility*, the goal is to construct an on-line scheduling algorithm that has the largest worst-case average utility possible. Recall the notation $V(\rho^\sigma_{\mathcal{A}}, k)$ for the cumulative utility in the first k time slots of an on-line scheduling algorithm $\mathcal{A}$ with schedule $\rho^\sigma_{\mathcal{A}}$ under the released job sequence σ. Formally, the task is to construct an on-line scheduling algorithm $\mathcal{A}$ such that, for any online-scheduling algorithm $\mathcal{A}'$,

$$\inf_{\sigma \in \mathcal{J}} \liminf_{k \to \infty} \frac{1}{k} V(\rho^\sigma_{\mathcal{A}}, k) \geqslant \inf_{\sigma \in \mathcal{J}} \liminf_{k \to \infty} \frac{1}{k} V(\rho^\sigma_{\mathcal{A}'}, k),$$

   where $\mathcal{J}$ is the set of admissible job sequences.
2. In *competitive synthesis*, the task is to construct an on-line scheduling algorithm with the largest possible competitive ratio. That is, we are looking for an on-line algorithm $\mathcal{A}$ such that, for any on-line algorithm $\mathcal{A}'$, we have $\mathcal{CR}_{\mathcal{J}}(\mathcal{A}) \geqslant$

$\mathcal{CR}_{\mathcal{J}}(\mathcal{A}')$, where $\mathcal{CR}_{\mathcal{J}}(\mathcal{A})$ is the competitive ratio of algorithm $\mathcal{A}$ under the set $\mathcal{J}$ of admissible job sequences (see Eq. (1) in Sect. 2 for the definition of $\mathcal{CR}_{\mathcal{J}}$).

As in the competitive analysis case of Sect. 4, it suffices to consider only on-line scheduling algorithms encoded as LTSs (see Remark 1). In the following, we consider that $\mathcal{J} = \Sigma^{\omega}$, that is, there are no restrictions on the released job sequences. In Remark 7 below, we outline how the results can be extended to additional safety, liveness, and limit-average automata constraining $\mathcal{J}$ (see also Sect. 3.2). Finally, we conclude with a note on the *worst-case utility ratio*, namely the worst-case limiting average utility of the best online algorithm over the worst-case limiting average utility achievable by a clairvoyant algorithm (for possibly different job sequences).

5.3.1 Synthesis for worst-case average utility

Given a taskset, we can compute the worst-case average utility that can be achieved by any on-line scheduling algorithm. For this, we construct a *non-deterministic* finite-state LTS $L_{\mathcal{G}} = (S_{\mathcal{G}}, s_{\mathcal{G}}, \Sigma, \Pi, \Delta_{\mathcal{G}})$ with an associated reward function $r_{\mathcal{G}}$ that can simulate all possible on-line algorithms. Such an LTS has already been introduced in Sect. 3 for the clairvoyant algorithm. Note that the latter implements memoryless strategies, as all required history information is encoded in the state.

We can interpret such a non-deterministic LTS as a perfect-information graph game $\mathcal{G} = \langle S_{\mathcal{G}}, \Sigma_1, \Sigma_2, \delta \rangle$, where Σ_1 (the actions of Player 1) correspond to the output actions Π in $L_{\mathcal{G}}$, and Σ_2 (the actions of Player 2) correspond to the input actions Σ in $L_{\mathcal{G}}$. That is, Player 2 (the adversary) chooses the released tasks, while Player 1 chooses the actual transitions in δ actually taken.

Thus, we indeed have a perfect-information game, and every memoryless strategy for Player 1 corresponds to a scheduling algorithm and vice-versa (i.e., every scheduling algorithm is a memoryless strategy of Player 1 in the game $\mathcal{G}$). The weight function w for the mean-payoff objective of $\mathcal{G}$ is identical to the reward function $r_{\mathcal{G}}$, and the start state s^1 is the initial state $s_{\mathcal{G}}$ of $L_{\mathcal{G}}$. The worst-case utility of a given on-line algorithm, corresponding to a memoryless strategy $\pi \in \Pi^M_{\mathcal{G}}$, is hence

$$\inf_{\sigma \in \Sigma_{\mathcal{G}}} \mathsf{MP}(w, \mathscr{P}(s^1, \pi, \sigma))$$

and the worst-case utility of the optimal on-line algorithm is given by

$$\sup_{\pi \in \Pi^M_{\mathcal{G}}} \inf_{\sigma \in \Sigma_{\mathcal{G}}} \mathsf{MP}(w, \mathscr{P}(s^1, \pi, \sigma)). \tag{3}$$

Using the results of Theorem 2, we obtain the following theorem.

Theorem 4 *The following assertions hold:*

1. *Whether there exists an on-line algorithm with worst-case average utility at least v 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.*

2. *An on-line algorithm with optimal worst-case average utility can be constructed in time $O(|S_{\mathscr{G}}| \cdot m \cdot V_{\max})$, where $|S_{\mathscr{G}}|$ (resp. m) is the number of states (resp. transitions) of the non-deterministic LTS $L_{\mathscr{G}}$.*

5.3.2 Competitive synthesis

Given a taskset and a rational $\nu \in \mathbb{Q}$, the competitive synthesis problem asks to determine whether there exists an on-line scheduling algorithm that achieves a competitive ratio of at least ν, and to determine the optimal competitive ratio ν^*. Recall the non-deterministic LTS $L_{\mathscr{G}} = (S_{\mathscr{G}}, s_{\mathscr{G}}, \Sigma, \Pi, \Delta_{\mathscr{G}})$ and reward function $r_{\mathscr{G}}$ in the synthesis for the worst-case average utility. For solving the competitive synthesis problem, we construct a partial-observation game $\mathscr{G}_{\mathcal{CR}}$ as follows: $\mathscr{G}_{\mathcal{CR}} = \langle S_{\mathscr{G}} \times S_{\mathscr{G}}, \Sigma_1, \Sigma_2 \times \Sigma_1, \delta, \mathcal{O}_S, \mathcal{O}_\Sigma \rangle$, where $\Sigma_1 = \Pi$ and $\Sigma_2 = \Sigma$. Intuitively, we construct a product game with two components, where Player 1 only observes the first component (the on-line algorithm) and makes the choice of the transition $\alpha_1 \in \Sigma_1$ there; Player 2 is in charge of choosing the input $\alpha_2 \in \Sigma_2$ and also the transition $\alpha_1' \in \Sigma_1$ in the second component (the clairvoyant algorithm). However, due to partial observation, Player 1 does not observe the choice of the transitions of the clairvoyant algorithm.

Formally, the appropriate transition and the observation mapping are defined as follows:

(i) Transition function $\delta : (S_{\mathscr{G}} \times S_{\mathscr{G}}) \times \Sigma_1 \times (\Sigma_2 \times \Sigma_1) \to (S_{\mathscr{G}} \times S_{\mathscr{G}})$ with

$$\delta((s_1, s_2), \alpha_1, (\alpha_2, \alpha_1')) = (\delta(s_1, \alpha_1, \alpha_2), \delta(s_2, \alpha_1', \alpha_2)).$$

(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 action (α_2, α_1') of Player 2 to its first component α_2 as well (i.e., the input from Player 2), i.e., $\mathsf{obs}_\Sigma((\alpha_2, \alpha_1')) = \alpha_2$.

The two reward functions needed for solving the ratio objective in the game are defined as follows: the reward function w_1 gives reward according to $r_{\mathscr{G}}$ applied to the transitions of the first component. The reward function w_2 assigns the reward according to $r_{\mathscr{G}}$ applied to the transitions of the second component. Note that this construction ensures that we compare the utility of an on-line algorithm (transitions of the first component chosen by Player 1) and an off-line algorithm (chosen by Player 2 using the second component) that operate on the *same* input sequence.

It follows that an on-line algorithm with competitive-ratio at least ν exists iff $\sup_{\pi \in \Pi^M_{\mathscr{G}}} \inf_{\sigma \in \Sigma_{\mathscr{G}}} \mathsf{Ratio}(w_1, w_2, \mathscr{P}(s^1, \pi, \sigma)) \geqslant \nu$, where $s^1 = (s_{\mathscr{G}}, s_{\mathscr{G}})$ is the start state derived from the LTS $L_{\mathscr{G}}$. By Theorem 3, the decision problem is in NP in the size of the game $\mathscr{G}_{\mathcal{CR}}$. Since the strategy of Player 1 can directly be translated to an on-line scheduling algorithm, the solution of the synthesis problem follows from the witness strategy for Player 1. We hence obtain:

Theorem 5 *For the class of scheduling problems defined in Sect. 2, the decision problem of whether there exists an on-line scheduler with a competitive ratio at least*

a rational number ν is in NP *in the size of the LTS constructed from the scheduling problem.*

Finally, finding the optimal competitive ratio ν^* (and a scheduling algorithm ensuring it) is possible by searching for $\sup\{\nu \in \mathbb{Q} :$ the answer to the decision problem is yes$\}$.

Remark 7 Using the reduction of Theorem 5 together with Remark 6, we obtain that the competitive synthesis problem in the presence of safety, liveness, and limit-average constraints specified as constrained automata is in NP in the size of the synchronous product of the corresponding LTSs.

5.3.3 Synthesis for worst-case utility ratio

We conclude our considerations regarding synthesis with the worst-case utility ratio problem, namely, determining the worst-case limiting average utility of the best online algorithm over the worst-case limiting average achievable by a clairvoyant algorithm. In sharp contrast to the competitive ratio, the job sequences used by the on-line and off-line algorithm for computing this utility ratio may be different. Formally, we are interested in determining an online scheduling algorithm $\mathcal{A}$ that maximizes the following expression:

$$\mathcal{UR} = \liminf_{k \to \infty} \frac{\inf_{\sigma \in \mathcal{J}} V(\rho^{\sigma}_{\mathcal{A}}, k)}{\inf_{\sigma \in \mathcal{J}} V(\rho^{\sigma}_{\mathcal{C}}, k)}. \tag{4}$$

The numerator of $\mathcal{UR}$ corresponds to the synthesis for the worst case average utility problem, whose solution is given by Eq. (3) in the respective game. Similarly, the denominator is given by the following objective in the same game:

$$\inf_{\sigma \in \Pi_{\mathcal{G}}} \sup_{\pi \in \Sigma_{\mathcal{G}}} \mathsf{MP}(w, \mathscr{P}(s^1, \pi, \sigma)).$$

Herein, the input sequence is fixed (by choosing a strategy for Player 1) *before* the job sequence is fixed (by choosing a strategy for Player 2, possibly non-memoryless). According to the determinacy guaranteed by Theorem 2, Eqs. (3) and (5.3.3) are equal, hence $\mathcal{UR} = 1$: the worst case average utility of the optimal online and the clairvoyant algorithm coincide.

Remark 8 (Complexity with respect to the taskset) Theorem 4 and Theorem 5 establish complexity upper bounds for the synthesis for worst-case utility, and competitive synthesis problems as a function of the size of the non-deterministic LTS $L_{\mathcal{G}}$. In general, the size of $L_{\mathcal{G}}$ is exponential in the bit representation of the taskset $\mathcal{T}$. Hence, if the input to our algorithms is the taskset $\mathcal{T}$, the polynomial upper bounds of Theorem 4 and Theorem 5 translate to exponential upper bounds in the size of $\mathcal{T}$.

Remark 9 (Memory of the synthesized scheduler) The memory-space requirement of the synthesized scheduler is upper-bounded by $O(C_{\max}^{N \cdot (D_{\max}-1)})$, where N is the

number of tasks, $D_{\max}$ the maximum task deadline and $C_{\max}$ is the maximum execution time. This holds since the state of the online scheduler is an $N \times (D_{\max} - 1)$ matrix, where each entry of the matrix denotes the remaining execution time of a job.

6 Conclusions

We presented a flexible framework for the automated competitive analysis and competitive synthesis of real-time scheduling algorithms for firm-deadline tasksets using graph games. For competitive analysis, scheduling algorithms are specified as (deterministic) labeled transition systems. The rich formalism of automata on infinite words is used to express optional safety, liveness and limit-average constraints in the generation of admissible job sequences. Our prototype implementation uses an optimized reduction of the competitive analysis problem to the problem of solving certain multi-objectives in multi-graphs. Our comparative experimental case study demonstrates that it allows to solve small-sized tasksets efficiently. Moreover, our results clearly highlight the importance of a fully automated approach, as there is neither a "universally" optimal algorithm for all tasksets (even in the absence of additional constraints) nor an optimal algorithm for different constraints in the same taskset. For competitive synthesis, we introduced a partial observation game with mean-payoff and ratio objectives. We determined the complexity of this game, and showed that it can be used to solve the competitive synthesis problem.

Future work will be devoted to adding additional constraints to the scheduling algorithms, like energy constraints. In order to scale-up to larger tasksets, we will also investigate advanced techniques for further reducing the size of the underlying (game) graphs. Finally, the core computational step in our framework is that of computing mean-payoff objectives in the underlying graphs. Developing faster algorithms for mean-payoff objectives for special graphs is an active area of research (Chatterjee et al. 2015; Chatterjee et al. 2014). Whether the structure of our graphs can be exploited to yield faster algorithms for our framework is an interesting future direction.

Acknowledgements This work has been supported by the Austrian Science Foundation (FWF) under the NFN RiSE (S11405 and S11407), FWF Grant P23499-N23, ERC Start Grant (279307: Graph Games), and Microsoft Faculty Fellows Award.

References

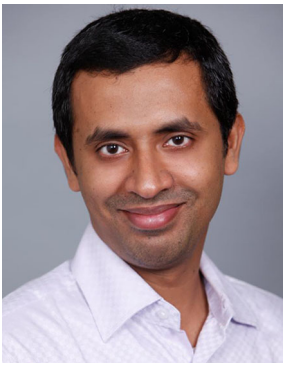
Abeni L, Buttazzo G (1998) Integrating multimedia applications in hard real-time systems. In: Proceedings of the 19th IEEE real-time systems symposium, RTSS '98, pp 4–13. https://doi.org/10.1109/REAL.1998.739726

Altisen K, Gößler G, Sifakis J (2002) Scheduler modeling based on the controller synthesis paradigm. Real-Time Syst. 23(1–2):55–84

Aydin H, Melhem R, Mossé D, Mejía-Alvarez P (2004) Power-aware scheduling for periodic real-time tasks. IEEE Trans Comput 53(5):584–600. https://doi.org/10.1109/TC.2004.1275298

Baruah SK, Haritsa JR (1997) Scheduling for overload in real-time systems. IEEE Trans Comput 46:1034–1039. https://doi.org/10.1109/12.620484
Baruah SK, Hickey ME (1998) Competitive on-line scheduling of imprecise computations. IEEE Trans Comput 47(9):1027–1032
Baruah S, Koren G, Mishra B, Raghunathan A, Rosier L, Shasha D (1991) On-line scheduling in the presence of overload. In: Proceedings of the 32nd annual symposium on foundations of computer science, FOCS '91, pp 100–110. https://doi.org/10.1109/SFCS.1991.185354
Baruah S, Koren G, Mao D, Mishra B, Raghunathan A, Rosier L, Shasha D, Wang F (1992) On the competitiveness of on-line real-time task scheduling. Real-Time Syst 4(2):125–144
Berkelaar M, Eikland K, Notebaert P (2004) lpsolve: open source (mixed-integer) linear programming system. Version 5.0.0.0
Bonifaci V, Marchetti-Spaccamela A (2012) Feasibility analysis of sporadic real-time multiprocessor task systems. Algorithmica 63(4):763–780. https://doi.org/10.1007/s00453-011-9505-6
Borodin A, El-Yaniv R (1998) Online computation and competitive analysis. Cambridge University Press, Cambridge
Brim L, Chaloupka J, Doyen L, Gentilini R, Raskin JF (2011) Faster algorithms for mean-payoff games. Form Methods Syst Des 38(2):97–118
Chatterjee K, Kößler A, Schmid U (2013) Automated analysis of real-time scheduling using graph games. In: Proceedings of the 16th ACM international conference on hybrid systems: computation and control, HSCC '13, pp 163–172. ACM, New York
Chatterjee K, Henzinger M, Krinninger S, Nanongkai D (2014) Polynomial-time algorithms for energy games with special weight structures. Algorithmica 70(3):457–492
Chatterjee K, Pavlogiannis A, Kößler A, Schmid U (2014) A framework for automated competitive analysis of on-line scheduling of firm-deadline tasks. In: RTSS '14, pp 118–127
Chatterjee K, Ibsen-Jensen R, Pavlogiannis A (2015) Faster algorithms for quantitative verification in constant treewidth graphs. In: CAV
Clarke EM Jr, Grumberg O, Peled DA (1999) Model checking. MIT Press, Cambridge
Cruz RL (1991) A calculus for network delay. I. Network elements in isolation. IEEE Trans Inf Theory 37(1):114–131. https://doi.org/10.1109/18.61109
Dertouzos ML (1974) Control robotics: The procedural control of physical processes. In: IFIP Congress, pp 807–813
Devadas V, Li F, Aydin H (2010) Competitive analysis of online real-time scheduling algorithms under hard energy constraint. Real-Time Syst 46(1):88–120. https://doi.org/10.1007/s11241-010-9100-y
Ehrenfeucht A, Mycielski J (1979) Positional strategies for mean payoff games. Int J Game Theory 8(2):109–113
Englert M, Westermann M (2007) Considering suppressed packets improves buffer management in qos switches. In: Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms, SODA 2007, New Orlean, January 7–9, 2007, pp 209–218. http://dl.acm.org/citation.cfm?id=1283383.1283406
Galperin H, Wigderson A (1983) Succinct representations of graphs. Inf Control 56(3):183–198
Golestani SJ (1991) A framing strategy for congestion management. IEEE J Sel Areas Commun 9(7):1064–1077. https://doi.org/10.1109/49.103553
Gupta BD, Palis MA (2001) Online real-time preemptive scheduling of jobs with deadlines on multiple machines. J Sched 4(6):297–312. https://doi.org/10.1002/jos.85
Haritsa JR, Carey MJ, Livny M (1990) On being optimistic about real-time constraints. In: Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, PODS '90, pp 331–343. ACM, New York. https://doi.org/10.1145/298514.298585
Karp RM (1972) Reducibility among combinatorial problems. In: Proceedings of the Complexity of computer computations. Springer US
Karp RM (1978) A characterization of the minimum cycle mean in a digraph. Discret Math 23(3):309–311
Khachiyan LG (1979)) A polynomial algorithm in linear programming. Dokl Akad Nauk SSSR 244:1093–1096
Koren G, Shasha D (1995) D^{over}: an optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. SIAM J Comput 24(2):318–339. https://doi.org/10.1137/S0097539792236882
Koutsoupias E (2011) Scheduling without payments. In: Proceedings of the 4th international conference on algorithmic game theory, SAGT'11, pp 143–153. Springer, Berlin, Heidelberg

Leung JT (1989) A new algorithm for scheduling periodic, real-time tasks. Algorithmica 4(1–4):209–219. https://doi.org/10.1007/BF01553887
Locke CD (1986) Best-effort decision-making for real-time scheduling. Ph.D. thesis, CMU, Pittsburgh
Lu Z, Jantsch A (2007) Admitting and ejecting flits in wormhole-switched networks on chip. IET Comput Digit Tech 1(5):546–556. https://doi.org/10.1049/iet-cdt:20050068
Lübbecke E, Maurer O, Megow N, Wiese A (2016) A new approach to online scheduling: approximating the optimal competitive ratio. ACM Trans Algorithms 13(1):15:1–15:34
Madani O (2002) Polynomial value iteration algorithms for deterministic MDPs. In: Proceedings of the 18th conference on uncertainty in artificial intelligence, UAI '02, pp 311–318
Martin DA (1975) Borel determinacy. Ann Math 102(2): 363–371 . http://www.jstor.org/stable/1971035
Nisan N, Roughgarden T, Tardos E, Vazirani VV (2007) Algorithmic game theory. Cambridge University Press, New York
Palis MA (2004) Competitive algorithms for fine-grain real-time scheduling. In: Proceedings of the 25th IEEE real-time systems symposium, RTSS '04, pp 129–138. IEEE Computer Society. http://doi.ieeecomputersociety.org/10.1109/REAL.2004.14
Papadimitriou CH (1993) Computational complexity. Addison-Wesley, Reading
Porter R (2004) Mechanism design for online real-time scheduling. In: Proceedings of the 5th ACM conference on electronic commerce, EC '04, pp 61–70. ACM, New York. https://doi.org/10.1145/988772.988783
Rajkumar R, Lee C, Lehoczky J, Siewiorek D (1997) A resource allocation model for qos management. In: Proceedings of the 18th IEEE real-time systems symposium, RTSS '97, pp 298 –307. https://doi.org/10.1109/REAL.1997.641291
Shapley LS (1953) Stochastic games. Proc Natl Acad Sci USA 39(10):1095–1100. http://www.pnas.org/content/39/10/1095.short
Sheikh AA, Brun O, Hladik PE, Prabhu BJ (2011) A best-response algorithm for multiprocessor periodic scheduling. In: Proceedings of the 2011 23rd euromicro conference on real-time systems, ECRTS '11, pp 228–237. IEEE Computer Society, Washington. https://doi.org/10.1109/ECRTS.2011.29
Tarjan R (1972) Depth-first search and linear graph algorithms. SIAM J Comput 1(2):146160
Velner Y, Chatterjee K, Doyen L, Henzinger TA, Rabinovich A, Raskin JF (2015) The complexity of multi-mean-payoff and multi-energy games. Inf Comput 241(Suppl C):177–196. https://doi.org/10.1016/j.ic.2015.03.001
Zwick U, Paterson M (1996) The complexity of mean payoff games on graphs. Theoret Comput Sci 158(12):343–359



Krishnendu Chatterjee is Professor at IST Austria. He obtained B.Tech (Hons) in Computer Science from IIT Kharagpur, MS and PhD from UC Berkeley. His research interests include logic and automata theory, graph games and its applications, and evolutionary game theory, and co-authored numerous papers in the above research areas. He won the President of India Gold Medal, Ackermann Award, Microsoft Faculty Fellows Award, and ERC Start Grant.

Andreas Pavlogiannis is a postdoctoral researcher at EPFL. He obtain a Computer Engineer Diploma from the University of Patras, a Master of Science from UC Davis, and Ph.D. from IST Austria. His research interests revolve around the algorithmic and mathematical analysis of systems. His primary line of work is on programming languages and formal verification of software, with an emphasis on quantitative and concurrency aspects.

Alexander Kößler Photography and Biography unavailable.

Ulrich Schmid is full professor and head of the Embedded Computing Systems Group at the Institutfür Technische Informatik at TU Vienna. He studied computer science and mathematics and also spent several years in industrial electronics and embedded systems design. He authored and co-authored numerous papers in the field of theoretical and technical computer science and received several awards and prices, like the Austrian START-price 1996. His current research interests focus on the mathematical analysis of fault-tolerant distributed algorithms and real-time systems, with special emphasis on their application in systems-on-chips and networked embedded systems.