HAL
archives-ouvertes.fr

Assigning sporadic tasks to unrelated machines

Alberto Marchetti-Spaccamela, Cyriel Rutten, Suzanne Van Der Ster, Andreas Wiese

▶ **To cite this version:**

Alberto Marchetti-Spaccamela, Cyriel Rutten, Suzanne Van Der Ster, Andreas Wiese. Assigning sporadic tasks to unrelated machines. Mathematical Programming, Springer Verlag, 2015, 152 (1-2), pp.247-274 <10.1007/s10107-014-0786-9>. <hal-01249101>

HAL Id: hal-01249101
https://hal.inria.fr/hal-01249101

Submitted on 4 Jul 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Assigning sporadic tasks to unrelated machines

Alberto Marchetti-Spaccamela · Cyriel Rutten · Suzanne van der Ster · Andreas Wiese

Abstract We study the problem of assigning sporadic tasks to unrelated machines such that the tasks on each machine can be feasibly scheduled. Despite its importance for modern real-time systems, this problem has not been studied before. We present a polynomial-time algorithm which approximates the problem with a constant speedup factor of $8 + 2\sqrt{6} \approx 12.9$ and show that any polynomial-time algorithm needs a speedup factor of at least 2, unless $P = NP$. In the case of a constant number of machines we give a polynomial-time approximation scheme. Key to these results are two new relaxations of the demand bound function, the function that yields a sufficient and necessary condition for a task system on a single machine to be feasible. In particular, we present new methods to approximate this function to obtain useful structural properties while incurring only bounded loss in the approximation quality. For the constant speedup result we employ a very general rounding procedure for linear programs (LPs) which model assignment problems with capacity-type constraints. It ensures that the cost of the rounded integral solution is no more than the cost of the optimal fractional LP solution and the capacity constraints are violated only by a bounded factor, depending on the structure of the matrix that defines the LP. In fact,

A. Marchetti-Spaccamela
Sapienza University of Rome, Rome, Italy
e-mail: alberto@dis.uniroma1.it

C. Rutten
Maastricht University, Maastricht, The Netherlands
e-mail: cyrielrutten@gmail.com

S. van der Ster (✉)
Vrije Universiteit, Amsterdam, The Netherlands
e-mail: suzanne.vander.ster@vu.nl

A. Wiese
Max-Planck Institut für Informatik, Saarbrücken, Germany
e-mail: awiese@mpi-inf.mpg.de

our rounding scheme generalizes the well-known 2-approximation algorithm for the generalized assignment problem due to Shmoys and Tardos.

Keywords Scheduling · Sporadic task systems · Unrelated machines · Demand bound function · Integer linear programming · Rounding

1 Introduction

The *sporadic task model* is a model of recurrent processes in hard real-time systems that has received great attention in the last years; see for example [7,12], and references therein. A sporadic task $\tau = (c_\tau, d_\tau, t_\tau)$ is characterized by a worst-case execution time c_τ, a relative deadline d_τ, and a minimum interarrival separation t_τ. Such a sporadic task generates a potentially infinite sequence of jobs with successive job arrivals separated by at least t_τ time units, it has an execution requirement less than or equal to c_τ and a deadline that occurs d_τ time units after its arrival time.

A sporadic task system is comprised of several such sporadic tasks. Since the actual interarrival times can vary, there are infinitely many job sequences that can be generated. A sporadic task system is said to be *feasible* upon a specified platform if it is possible to schedule the system on the platform such that all jobs of all tasks will meet their deadlines, under all permissible combinations of job arrival sequences by the different tasks comprising the system.

The feasibility analysis of sporadic task systems on single processors has been extensively studied [7]. It is known that the Earliest Deadline First (EDF) algorithm, that schedules at any time the job with the earliest absolute deadline, is optimal in the sense that for any sequence of jobs it produces a valid schedule, whenever a valid schedule exists [27]. However, it is co-*NP*-hard to decide whether a task system is feasible on a single machine [18].

On multiprocessor systems, there are two main paradigms for scheduling: *global* and *partitioned* scheduling. In the former, all tasks can use all machines, and jobs can even be migrated from one machine to another. In the partitioned scheduling approach each task has to be assigned to one of the machines such that all its jobs have to be executed on this specific machine. Since the process of partitioning tasks among processors reduces a multiprocessor scheduling problem to a series of single-processor problems, the optimality of EDF for preemptive single-processor scheduling makes EDF a reasonable algorithm to use as the run-time scheduling algorithm on each machine.

In recent years, hardware design has seen a highly visible trend towards heterogeneous processors. In particular, modern hardware architectures often contain specialized processors for certain tasks (e.g., graphical processors, floating-point units). To model the actual behavior of the different types of processors when making scheduling decisions, in this paper we assume that the given machines are *unrelated*; i.e., we assume that the processing time of each task depends on the machine where it is executed, including the possibility that some tasks cannot be executed on some machines at all.

For attacking our problem, we will be using an integer linear program (ILP). It is *NP*-hard to solve ILPs optimally [22], but the corresponding LP relaxation (i.e., where the integrality constraints on the variables are relaxed) can be solved in polynomial time. Hence, a common approach is to solve the relaxation and transform the obtained fractional solution into an integer solution, at the cost of worsening the objective value or violating the constraints in a controlled way. As part of our techniques, we present a new method to round a very general class of linear programs for assignment problems, without worsening the objective value and with carefully bounding the (then unavoidable) errors on the constraints due to the rounding.

Related work As mentioned above, deciding whether a task system is feasible on a single machine is a co-*NP*-hard problem [18]. This motivates approximate feasibility tests that run efficiently but introduce an error in the decision process, controlled by an accuracy parameter α (the speedup). If an *α-approximation test* returns "feasible", then the task set is guaranteed to be feasible on an α-speed processor(s); if the test returns "infeasible", the task set is guaranteed to be infeasible on a unit-speed processor(s). For the case of a single processor, an FPTAS feasibility test for EDF has been proposed [14] (i.e., for any $\epsilon > 0$, there exists a $(1+\epsilon)$-approximation test with running time polynomial in the number of tasks and in $1/\epsilon$).

In the case of m identical processors, assuming the global paradigm, the natural EDF-policy is no longer optimal, but it is known that any feasible collection of jobs on m machines of unit speed is schedulable using EDF on m machines of speed $2-\frac{1}{m}$ [28]. Also, a corresponding test for task sets is known [11,13]. Recently, Anand et al. [2] presented an online algorithm needing only a speedup factor of $e/(e-1) \approx 1.58$.

In the case of the partitioned paradigm, to the best of our knowledge, the multiple-machines case has only been studied for identical machines. Most of prior research has considered the special case of implicit-deadline systems in which all tasks have their deadlines equal to their period parameters (i.e., $d_\tau = t_\tau$ for all τ). Then, a set of tasks is feasible on one machine if and only if the sum of the utilizations c_τ/t_τ is at most one and the problem reduces to MAKESPAN MINIMIZATION. In Eisenbrand and Rothvoß [17] a PTAS was proposed for the case where tasks are scheduled according to fixed priorities using resource augmentation. In that paper also the existence of an FPTAS is ruled out, thus showing that the problem is strongly *NP*-hard.

If deadlines are not implicit, much less is known. Baruah and Fisher [9] propose an algorithm which can partition any set of tasks that is feasible on m machines such that the assignment is feasible if the machines run $4-\frac{2}{m}$ times faster. In Fisher et al. [19] a similar result is given if the tasks are scheduled according to static priorities, rather than with the more powerful EDF policy. Chen and Chakraborty [15] improved upon these results by showing that a deadline-monotonic policy with approximate demand bound functions leads to a feasibility test with speedup factor $\frac{3e-1}{e}-\frac{1}{m} \approx 2.6322-\frac{1}{m}$ in case of constrained deadlines ($d_\tau \leq t_\tau$ for all τ) and a speedup factor of $3-\frac{1}{m}$ for unconstrained deadlines.

As mentioned above, we assume the machines to be unrelated, meaning that the processing times of the tasks can differ on the different machines. To the best of our

knowledge the unrelated-machines setting has not been studied for arbitrary-deadline sporadic tasks. For implicit-deadline task systems some results are known, that are discussed below.

However, the setting with unrelated machines is well-studied for assigning jobs to minimize the makespan. Lenstra et al. [26] give a 2-approximation algorithm for the problem of minimizing the makespan of a set of jobs and prove that it is *NP*-hard to achieve a performance ratio better than 1.5. Despite a lot of effort, the only improvements in the setting of an arbitrary number of machines are a 1.75-approximation algorithm for the graph balancing case [16] and a $33/17 \approx 1.94$-estimation algorithm for the restricted assignment case [32]. A generalization of this problem is the Generalized Assignment Problem (GAP). In this problem, assigning a job j to a machine i incurs a certain (global) cost $c_{i,j}$. Shmoys and Tardos [31] give a 2-approximation for this problem; to be more precise, they devise an algorithm computing a schedule with makespan $2T$ and cost at most C, given that a solution with the same cost and a makespan of T exists.

Azar and Epstein [6] consider ℓ_p norms (for finite $p > 1$, rather than the makespan) for which they improved the previously best result of $\theta(p)$ [5] to a 2-approximation and even a $\sqrt{2}$-approximation for the ℓ_2 norm. This was improved to a better-than-two result for all $p > 1$ in [24].

For a constant number of machines, polynomial-time approximation schemes are known [20,26]. Rather than focusing on a constant number of machines, one could also consider the case of a constant number of machine types; each machine belongs to one of the types and the processing time of each job depends only on the job and the type of the machine it is assigned to. For the problem of assigning implicit-deadline task systems to unrelated machines of two different types, an approximation algorithm is given in [29] based on the first-fit heuristic with a worst-case performance ratio of 2. For the same problem, a PTAS was given later [30]. In the meantime, a PTAS was given [33] for any constant number of machine types. The authors justify their contribution in [30] by pointing out that their PTAS for two different types has a much lower run-time complexity than the one in [33] applied to two machine types.

The *NP*-hardness of solving arbitrary ILPs inspired the search for algorithms that transform the fractional solution of an LP relaxation into an integer solution. The goal is that such a solution either has a bounded loss on the objective function while satisfying all constraints, or gives a bound on the amount that any constraint is violated. There are many problems than can be well approximated using LP rounding, see e.g., [34] for an introduction.

The LP rounding procedure we will present in Sect. 3 produces an integral solution that maintains an objective value not worse than that of the optimal fractional solution, while guaranteeing a bound on the violation of any constraint in the system. This procedure is based on ideas presented in [23], where a solution for an integer linear program (without objective function) is found by rounding the solution to the LP relaxation such that each variable is rounded up or rounded down and each constraint of the program is violated by at most a constant factor (depending on the maximum sum of elements over all columns of the coefficient matrix).

Our contribution To the best of our knowledge, no non-trivial algorithm is known for assigning a set of sporadic tasks to a set of unrelated machines. We first present an algorithm that, given a task system for which a task assignment on m machines exists, finds a task assignment that can be scheduled on m machines that are $8+2\sqrt{6} \approx 12.9$ times as fast.

This result is obtained through a new rounding procedure that is first presented separately in Sect. 3. The procedure is designed for rounding a fractional solution of the LP relaxation of any assignment-type problem where items (e.g., tasks in our case) have to be assigned to resources (e.g., machines) while obeying some knapsack-type constraints. Given that each resource-item combination appears in at most γ constraints, the procedure preserves the constraint that every item is assigned and all other constraints are violated by at most a factor of $\gamma + 1$. Further, the cost of this rounded solution is no more than the optimal cost of the LP relaxation. In fact, the well-known 2-approximation algorithm for the generalized assignment problem by Shmoys and Tardos [31] can be derived as a direct corollary from our rounding procedure. We apply the procedure to a sparse LP formulation which approximately models the task assignment problem on unrelated machines and obtain the mentioned result (after some technical modifications of the LP). Also, we show that no polynomial-time algorithm can compute a task assignment needing a speedup factor of $2-\epsilon$ for $\epsilon > 0$, unless $P = NP$. Note that this bound is stronger than the best known $(3/2-\epsilon)$-hardness result for the contained problem of minimizing the makespan when scheduling jobs on unrelated machines [26].

For the case that the number of machines is fixed, we present a polynomial-time algorithm that either finds a feasible task assignment on m machines that are $1+\epsilon$ times as fast, or guarantees that no solution exists on unit-speed processors.

In order to be able to achieve these results, we need deep understanding of the *demand bound function* (dbf) which yields a necessary and sufficient condition for a task system to be feasible on one machine. In particular, we present two new relaxations for handling this well-studied function. For our result for an arbitrary number of machines, we give a set of sparse linear constraints which approximate the dbf up to a constant factor. Due to the sparsity we are able to design an efficient iterative rounding procedure.

For the case of a constant number of machines, we observe that we cannot exploit the technique of partitioning the task set into "big" and "small" tasks as in the job scheduling problem. A task having a small execution time or small utilization (formally defined in the next section) might still be very tight in the sense that its relative deadline is fairly small. Therefore, even if all large tasks are already assigned, assigning the small tasks is still very tricky, i.e., they cannot be scheduled easily by spreading them over machines which still have some capacity left. An important feature of our dbf relaxation is that the feasibility test of assigning a task with deadline D to a machine having tasks with deadlines $<D$ already assigned to it, requires only limited information of the previously assigned tasks. To be more explicit, the approximate demand bound function for each task only needs to be evaluated at a constant number of points. Afterwards we just approximate the function by the task's utilization. We exploit this feature and other tricks to polynomially bound the running time of a dynamic programming algorithm.

2 Preliminaries

Given is a set M of m parallel unrelated machines and a sporadic task system T, with $|T| = n$. Each task $\tau \in T$ is characterized by a set of values $(\{c_{i,\tau}\}_{i \in M}, d_\tau, t_\tau)$, where $c_{i,\tau}$ is its execution time on machine i, d_τ is its deadline, relative to its arrival time, and t_τ denotes the minimum interarrival time between two jobs of τ and is called the period. We assume all parameters to be integer and strictly positive. We study the problem of finding a task assignment $\mathcal{T} = \{\mathcal{T}_i\}_{i \in M}$ such that $\cup_i \mathcal{T}_i = T$. Without loss of generality we can restrict to task assignments with the property that $\mathcal{T}_i \cap \mathcal{T}_{i'} = \emptyset$ for any two machines $i \neq i'$.

A task assignment is *feasible* if for any machine i, any job arrival sequence of the tasks in $\mathcal{T}_i$ can be feasibly scheduled on i, allowing jobs to be preempted. Since each task is assigned to exactly one machine, after finding an assignment, EDF will be our scheduling algorithm of choice, by its optimality for single-machine scheduling [27].

An *α-approximation test* for the problem of assigning tasks to unrelated machines is an algorithm that runs in polynomial time and which either guarantees that there is no feasible integral assignment of the tasks to the given machines (running at unit speed), or finds an integral assignment which is feasible if the machines run at speed α.

By $u_{i,\tau}$ we denote the *utilization* of task τ on machine i and we define it as $u_{i,\tau} = c_{i,\tau}/t_\tau$. Given a task assignment $\mathcal{T}$, we define the utilization of each machine i by $u_i = \sum_{\tau \in \mathcal{T}_i} u_{i,\tau}$. Note that in a feasible assignment $u_i \leq 1$, for all $i \in M$, is a necessary but not sufficient condition for feasibility [27].

Clearly, the necessary condition $u_i \leq 1$ for all $i \in M$ implies that if there is a task τ such that $u_{i,\tau} > 1$, this task will never be assigned to machine i. Further, if for any task τ and machine i it holds that $c_{i,\tau} > d_\tau$, then τ will not be assigned to machine i. If it were assigned to machine i, the first job of τ clearly could not be completed before $c_{i,\tau}$ and would miss its deadline at d_τ.

Feasibility test The synchronous arrival sequence for task system T is defined to be the collection of job arrivals in which each task in T generates a job at time instant zero, and subsequent jobs arrive as soon as legally permitted (i.e., task τ generates a job at each time-instant $k\,t_\tau$, $k = 0, 1, 2, \ldots$).

It is known [10] that a set of sporadic tasks $\mathcal{T}_i$ is EDF-schedulable on machine i if and only if the following conditions are satisfied:

1. the utilization of the task system does not exceed 1, i.e., $u_i = \sum_{\tau \in \mathcal{T}_i} u_{i,\tau} \leq 1$,
2. all jobs with deadlines $[0, lcm_{\tau \in \mathcal{T}_i}(t_\tau)]$ in the synchronous arrival sequence of $\mathcal{T}_i$ meet their deadlines (where lcm denotes the least common multiple).

This immediately yields an exponential-time test to check whether $\mathcal{T}_i$ is EDF-schedulable; however we recall that the problem is co-*NP*-hard [18] and that it is not known whether it can be determined in pseudo-polynomial time.

A necessary and sufficient condition for a task system T to be schedulable is based on the *demand bound function*. In the case of unrelated machines we have the following condition.

Proposition 1 ([10]) *An assignment $\mathcal{T} = \{\mathcal{T}_i\}_{i \in M}$ is feasible for task system T if and only if for all $i \in M$*

$$dbf_{\mathcal{T},i}(s) := \sum_{\tau \in \mathcal{T}_i : d_\tau \leq s} \left\lfloor \frac{s + t_\tau - d_\tau}{t_\tau} \right\rfloor c_{i,\tau} \leq s \qquad \forall s \geq 0.$$

We write dbf_i instead of $dbf_{\mathcal{T},i}$ whenever the assignment $\mathcal{T}$ is clear from the context. Further, we define $dbf_i(\tau, s) := \lfloor (s + t_\tau - d_\tau)/t_\tau \rfloor c_{i,\tau}$, i.e., $dbf_i(\tau, s)$ denotes the contribution of task τ to $dbf_{\mathcal{T},i}(s)$.

3 Rounding procedure

In this section we introduce the LP rounding procedure that was designed for ILPs that represent assignment problems. After that, we show that this procedure generalizes the result of Shmoys and Tardos [31] for the Generalized Assignment Problem.

3.1 The rounding procedure

Almost any combinatorial optimization problem can be formulated as an integer linear programming (ILP) problem with integral (or binary) decision variables and linear constraints. A feasible or optimal solution to this ILP is then a feasible or optimal solution to the problem originally considered. Unfortunately, solving ILPs in general is *NP*-hard [22]. A common approach is to, instead, solve the corresponding *LP relaxation*, where the integrality constraints on the decision variables are relaxed. In a second step, one turns this fractional solution into an integral solution without (1) violating the constraints too much, or (2) losing too much in the objective value with respect to the LP optimum.

In this section we present an LP rounding procedure that is specially designed for ILPs that stem from assignment problems. It guarantees that after applying the rounding, the obtained integral solution violates the constraints from the ILP by only a bounded factor (depending on the structure of the LP matrix) and the objective value of the LP optimum is preserved. In Sect. 4 we use this procedure to obtain a constant-factor approximation test for the problem of assigning tasks to unrelated machines. Since the procedure itself is independent of the application we will use it for, we present it here in more general terminology than we will need later. The rounding procedure is quite similar to the procedure presented in [23], but in contrast to the latter, ensures that in the resulting integral solution one obtains a feasible integral assignment.

Assume we want to solve the following assignment problem: given is a set of n items and a set of m resources M and a set $R \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$ containing all combinations (i, j) such that it is allowed to assign item j to resource i. For each combination $(i, j) \in R$ we are given a cost $c_{i,j}$ denoting the cost of assigning item j to i. The goal is to assign each item j to a resource i to minimize the total cost, and such that for each resource i a set of knapsack-type constraints is satisfied. Each item appears in at most γ constraints per resource. W.l.o.g. we assume that each item-resource combination appears in *exactly* γ of these constraints. Formally, we are given the set R and for all $(i, j) \in R$ we define decision variable $z_{i,j}$, which is equal to 1 if item j is assigned to resource i, and 0 otherwise. Let $\theta := n\,\gamma$. We are also given

non-negative values $a_{i,j}^{\ell}$ and b_i^{ℓ} for all $(i, j) \in R$ and $\ell \in \{1, \ldots, \theta\}$, such that for each $(i, j) \in R$, $a_{i,j}^{\ell} > 0$ for at most γ out of the θ coefficients. We want to solve the following integer linear program (ILP).

$$\min \sum_{(i,j)\in R} c_{i,j} z_{i,j}$$

$$\sum_{i:(i,j)\in R} z_{i,j} = 1 \qquad \text{for } j = 1, \ldots, n; \tag{1a}$$

$$\sum_{j:(i,j)\in R} a_{i,j}^{\ell} z_{i,j} \leq b_i^{\ell} \qquad \text{for } i = 1, \ldots, m; \ell = 1, \ldots, \theta; \tag{1b}$$

$$z_{i,j} \in \{0, 1\} \qquad \text{for all } (i, j) \in R \tag{1c}$$

We denote by LP^0 the LP relaxation of the above ILP. The contribution of item j in constraint ℓ of resource i is expressed by coefficient $a_{i,j}^{\ell}$. Note that if this is larger than the 'capacity' b_i^{ℓ}, item j will never be assigned to resource i in an integral solution. However, for our reasoning in Sect. 4 we need to round LP relaxations of the above type where possibly $a_{i,j}^{\ell} > b_i^{\ell}$. We assume w.l.o.g. that $a_{i,j}^{\ell} \leq 1$ for all i, j, ℓ (which can be ensured by linear scaling).

Solving the LP relaxation of this problem usually does not give an integral solution and hence items are fractionally assigned to multiple resources. Our rounding procedure turns this solution into an integral solution, such that each capacity constraint is not violated by more than γ times the maximum coefficient on the left-hand side. Further, the cost of this rounded solution will not be more than the optimal cost from the fractional LP solution.

In typical assignment-type problems, the number of constraints where the same resource-item combination appears, is often low. For example, in the linear program that we will use in Sect. 4 for the problem of assigning tasks to unrelated machines, we will have that $\gamma = 2$. In an LP relaxation for the problem of scheduling jobs on unrelated machines to minimize the makespan, we even have $\gamma = 1$.

Our rounding procedure is iterative and in each iteration h we compute an extreme-point solution $\mathbf{z}^h$ of a linear program LP^h, where, as defined above, LP^0 equals the LP relaxation of the original assignment problem and for $h \geq 0$ each LP^{h+1} is obtained by fixing the value for some variable(s) or removing some constraint(s) of LP^h. Whenever during this process constraints have become redundant (e.g., because all their variables have been already fixed) we remove them from the LP.

Given a feasible fractional solution $\mathbf{z}^h$, to obtain LP^{h+1} we first fix all variables which are integral in $\mathbf{z}^h$, i.e., those variables are not allowed to be changed anymore in the remainder of the procedure. Let s be the number of variables in LP^h and let r_a and r_b be the number of constraints of types (1a) and (1b), respectively. Let $r = r_a + r_b$. To obtain LP^{h+1}, we either fix and delete one or more variables (in case that $s > r$), or delete a constraint while ensuring that in the final solution that constraint will not be violated too much. Later in this section we will explain how to identify these constraints and bound the amount that these constraints can be violated in the final solution. Along the way we ensure that the constraints of type (1a) are always satisfied exactly and that the costs in the objective are not increased.

ALGORITHM 1: Iterative Rounding Procedure

Input: A set of m resources, a set of n items and a set $R \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$ and for each $(i, j) \in R$ and $\ell \in \{1, \ldots, \theta\}$ values $c_{i,j} \geq 0$, $b_i^\ell \geq 0$ and $a_{i,j}^\ell \geq 0$ and LP^0, the LP relaxation of (1)
loop
  Solve LP^h consisting of s variables and r constraints and find extreme-point solution $\mathbf{z}^h$.
  if $s > r$ **then**
    $\mathbf{z}^h$ has at least one integral entry.
    fix all integral entries at their value, remove them from the program and update right-hand side coefficients
    remove all inequalities that have become redundant by fixing the integral values
  else
    find a resource i and $\ell \in \{1, \ldots, \theta\}$ such that $\max_{\mathbf{w} \in S} \sum_{j=1}^n a_{i,j}^\ell (w_{i,j} - z_{i,j}^h) \leq \gamma$, where $S = \{0, 1\}^s$
    remove the inequality corresponding to the found i and ℓ.
  end if
  The remaining linear program is LP^{h+1}
end loop
return Integral solution $\bar{z}_{i,j}$ for all $(i, j) \in R$ such that $c^\top \bar{\mathbf{z}} \leq c^\top \mathbf{z}^0$ and (5) holds.

Note that if there is some variable $z_{i,j}$ that is fixed at value 1 and removed from the program, for all $i' \in M \backslash \{i\}$, $z_{i',j}$ will be set to 0 and also be removed from the program. The constraint of type (1a) corresponding to this item j is then superfluous and will also be removed. We also update the right-hand side of each constraint where $z_{i,j}$ appears, taking into account that we decided to assign j to i which thus reduces the remaining capacities of i, for each of the up to θ remaining constraints.

The following lemma is the key to show that this algorithm works correctly.

Lemma 1 *Let LP^h be the linear program that is solved in iteration h of the rounding procedure, having s variables and r constraints. Let $\mathbf{z}^h$ be an extreme-point solution to this LP. If $s > r$, then $\mathbf{z}^h$ has at least one integral entry. If $s \leq r$, then there is a resource i and some $\ell \in \{1, \ldots, \theta\}$ such that $\max_{\mathbf{w} \in S} \sum_{j=1}^n a_{i,j}^\ell (w_{i,j} - z_{i,j}^h) \leq \gamma$, where S is the integer solution space for all remaining variables, i.e., $S = \{0, 1\}^s$.*

Proof Assume that the subproblem in iteration h (described by LP^h) is defined as $\mathbf{Az} \leq \mathbf{b}$. First assume that $s > r$. Then the null space of $\mathbf{A}$ is non-empty. Let $\mathbf{z}_0$ be a vector in the null space of $\mathbf{A}$. Since $\mathbf{z}^h$ is an extreme-point solution to LP^h it cannot be expressed as the convex combination of two (or more) solutions to LP^h (see also e.g., [21]). If $\mathbf{z}^h$ does not have any integral entry then we can find a value $\delta > 0$ such that $\mathbf{z}^h + \delta \cdot \mathbf{z}_0$ and $\mathbf{z}^h - \delta \cdot \mathbf{z}_0$ are both solutions to LP^h and, in particular, $\mathbf{z}^h$ is a convex combination of these two solutions. Therefore, $\mathbf{z}^h$ must have at least one integral entry.

Now assume that $s \leq r$. For this case, we show that there always exists a constraint l of type (1b) such that $\max_{\mathbf{w} \in S} \{(\mathbf{Aw})_l - (\mathbf{Az})_l\} \leq \gamma$. We show the statement by contradiction. Assume that the statement is not true; that is, for each constraint l of type (1b) it holds that there exists a vector $\mathbf{w} \in S$ such that

$$(\mathbf{Aw})_l - (\mathbf{Az})_l > \gamma. \tag{2}$$

In each previous round, if variables were removed from the program, also constraints that had become redundant, were removed. Therefore, for all variables present in the linear program in this round, the corresponding constraint of type (1a) is also still present in the linear program (and this constraint is not present if all of its variables have been removed from the program). It follows that

$$\sum_{i=1}^{m} \sum_{j:z_{i,j} \text{ present}} z_{i,j} = r_a. \tag{3}$$

Define L as the set of constraints of type (1b) present in the current linear program. For any $q = (i, j)$, let L^q denote the set of these constraints containing variable z_q. Then,

$$\begin{aligned}
\gamma(r - r_a) &= \gamma\, r_b \\
&\overset{(2)}{<} \sum_{l \in L} \max_{\mathbf{w} \in S} ((\mathbf{A}\mathbf{w})_l - (\mathbf{A}\mathbf{z})_l) \\
&\overset{\text{all } a_{l,q} \geq 0}{=} \sum_{l \in L} ((\mathbf{A}\mathbf{1})_l - (\mathbf{A}\mathbf{z})_l) \\
&= \sum_{l \in L} \sum_{q} a_{l,q}(1 - z_q) \\
&= \sum_{q} \sum_{l \in L^q} a_{l,q}(1 - z_q) \\
&\leq \sum_{q} \gamma(1 - z_q) \\
&= \gamma s - \sum_{q} \gamma z_q \\
&= \gamma(s - r_a).
\end{aligned} \tag{4}$$

The second inequality follows since each variable z_q appears in at most γ constraints of type (1b) and since we assumed that $a_{l,q} \leq 1$ for all constraints $l \in L$ and all variables q.

The chain of inequalities implies that $\gamma(r - r_a) < \gamma(s - r_a) \Rightarrow r < s$ which is a contradiction to being in the case that $s \leq r$. Hence, we conclude that if $s \leq r$, there must be a constraint l of type (1b), for which $\max_{\mathbf{w} \in S}\{(\mathbf{A}\mathbf{w})_l - (\mathbf{A}\mathbf{z})_l\} \leq \gamma$. □

Given an extreme-point solution $\mathbf{z}^h$ for a linear program LP^h, we can find in polynomial time a constraint which can be dropped, assuming that the number of variables is bounded by the number of constraints (i.e., $s \leq r$). For each of the polynomial number of constraints, we try the most "pessimistic" vector $\mathbf{w} \in S = \{0, 1\}^s$. This vector $\mathbf{w}$ is obtained by taking $w_{i,j} = 1$ for all (i, j), since $a^{\ell}_{i,j} \geq 0$ for all (i, j) and all $\ell \in \{1, \ldots, \theta\}$.

Theorem 1 *Let $m, n, \gamma \in \mathbb{N}$. Let $\theta = n\gamma$. Suppose we are given a set $R \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$ of pairs (i, j) that we define a decision variable $z_{i,j}$ for. For each $(i, j) \in R$ and $\ell \in \{1, \ldots, \theta\}$ we are given values $c_{i,j} \geq 0$, $b^{\ell}_{i} \geq 0$, and*

$a^{\ell}_{i,j} \geq 0$, *such that for each* $(i, j) \in R$, $a^{\ell}_{i,j} > 0$ *for at most* γ *coefficients. Assume that the resulting linear program* LP^0, *as given in* (1), *is feasible and denote by* $\mathbf{z}^*$ *its optimal solution. Then there is a polynomial-time algorithm computing an integral solution* $\bar{\mathbf{z}}$ *with* $c^\top \bar{\mathbf{z}} \leq c^\top \mathbf{z}^*$ *which satisfies*

$$\sum_{i:(i,j)\in R} \bar{z}_{i,j} = 1 \qquad for\, j = 1, \ldots, n; \tag{5a}$$

$$\sum_{j:(i,j)\in R} a^{\ell}_{i,j} \bar{z}_{i,j} \leq b^{\ell}_i + \gamma \max_j a^{\ell}_{i,j} \qquad for\, i = 1, \ldots, m; \ell = 1, \ldots, \theta; \tag{5b}$$

$$\bar{z}_{i,j} \in \{0, 1\} \qquad for\ all\ (i, j) \in R. \tag{5c}$$

Proof Before applying the rounding procedure, we scale each linear constraint such that $\max_j a^{\ell}_{i,j} = 1$ for each combination of a resource i and a value $\ell \in \{1, \ldots, \theta\}$.

We apply our iterative rounding procedure, where LP^0 equals the LP relaxation of (1) and for each $h \geq 0$, LP^{h+1} is obtained from LP^h by either fixing the value of some variable(s) or removing some constraint(s). Suppose we computed an extreme-point solution $\mathbf{z}^h$ for LP^h. Then if $s > r$, according to Lemma 1, $\mathbf{z}^h$ must have at least one integral value. Those variables that have an integer value in $\mathbf{z}^h$ are fixed at these values and the variables are removed from the program. If a variable $z_{i,j}$ is fixed at value 1, then for all $i' \in M \backslash \{i\}$, the variables $z_{i',j}$ will be fixed at value 0 and be removed and the constraint of type (1a) corresponding to j will also be removed. Also the right-hand side of each constraint of type (1b) where $z_{i,j}$ appears will be updated, considering that assigning item j to resource i used $a^{\ell}_{i,j}$ of the capacity in constraint ℓ. Note that the optimal objective value of the resulting linear program LP^{h+1} is not larger than the optimal objective in LP^h since $\mathbf{z}^h$ induces a solution to LP^{h+1} with the same objective value as $\mathbf{z}^h$ gives in LP^h.

If $s \leq r$, we know by Lemma 1 that a constraint l exists, such that

$$\max_{\mathbf{w}\in S} \{(\mathbf{A}\mathbf{w})_l - (\mathbf{A}\mathbf{z})_l\} \leq \gamma \tag{6}$$

and it can be found in polynomial time. To this end, we check for each resource i whether $\sum_j (1 - z_{i,j}) \leq \gamma$. This is sufficient since all $a^{\ell}_{i,j} \geq 0$ and the maximum value any variable $z_{i,j}$ can take is 1. (In fact, it is even true that if the latter condition is satisfied *all* constraints of a given resource can be removed.) Since the removed constraint satisfies (6), we know that whatever value the variables in this constraint will get in the final solution, the additive error in the right-hand side of this constraint will be at most γ, assuming that all $a^{\ell}_{i,j} \leq 1$. Note that removing constraints cannot increase the costs of the final solution. At the end we scale all constraints back to their original values. Thus, the resulting solution satisfies (5b).

Finally, we know that in each iteration either of the two cases of Lemma 1 applies and thus after a polynomial number of iterations either all constraints or all variables are removed and we are done. □

3.2 Generalized assignment problem

In fact, a direct consequence of Theorem 1 is a 2-approximation algorithm for the Generalized Assignment Problem (GAP), as it was first presented by Shmoys and Tardos [31]. Given are a set of n jobs and a set of m unrelated machines. For each combination of a job j and a machine i we are given the running time $p_{i,j}$ of j on i and a cost $c_{i,j}$. For given values $C \geq 0$ and $T \geq 0$, the goal is to find an assignment of the jobs to the machines with total cost at most C and such that each machine has a makespan of at most T. In Shmoys and Tardos [31], a polynomial-time algorithm is presented that finds a solution with cost at most C and makespan at most $2T$, given that a solution of cost at most C and makespan at most T exists. The canonical LP formulation for GAP is the following:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{i,j} x_{i,j}$$

$$\sum_{i=1}^{m} x_{i,j} = 1 \qquad \text{for } j = 1, \ldots, n; \tag{7a}$$

$$\sum_{j=1}^{n} p_{i,j} x_{i,j} \leq T \qquad \text{for } i = 1, \ldots, m; \tag{7b}$$

$$x_{i,j} \geq 0 \qquad \text{for all } (i, j) \text{ s.t. } p_{i,j} \leq T \tag{7c}$$

If the optimal objective value is strictly greater than C then there is no solution obeying C and T. Otherwise, Theorem 1 directly gives a polynomial-time procedure that finds an integral solution with cost at most C and a makespan of at most $2T$, using that we have exactly one constraint per machine, i.e., $\gamma = 1$. In fact, for this specific LP it is known that the iterative rounding procedure described above gives the mentioned result; see e.g., [25].

4 Arbitrary number of machines

In this section we present an $\alpha = 8 + 2\sqrt{6} \approx 12.9$-approximation test for assigning tasks to unrelated machines and we show that the problem is *NP*-hard to approximate to within a ratio of $2 - \epsilon$ for any $\epsilon > 0$.

4.1 Constant-factor approximation test

We will formulate the problem of assigning tasks to unrelated machines as a linear program, such that the tasks on each machine can be feasibly scheduled using the EDF-scheduler. First, we derive a set of linear inequalities which are

– necessary, meaning that they are fulfilled by any feasible assignment,
– approximately sufficient, meaning that any integral assignment which (approximately) fulfills the constraints is feasible if the speed of the machine is increased by some constant factor, and
– sparse, meaning that each variable occurs in only two capacity constraints.

For each pair of a machine i and a task τ, such that $u_{i,\tau} \leq 1$ and $c_{i,\tau} \leq d_\tau$, we introduce a variable $y_{i,\tau}$ modeling to assign τ to machine i. Note that for pairs (i, τ) that do not satisfy these conditions we do not need a variable $y_{i,\tau}$ since it will be infeasible to assign such a τ to machine i. The first constraints are utilization bounds on all tasks assigned to the same machine i. Formally, we demand that

$$\sum_{\tau \in T} u_{i,\tau} y_{i,\tau} \leq 1 \qquad \forall i \in M. \tag{8}$$

Secondly, we require that for all tasks with their deadline in the interval $(2^{k-1}, 2^k]$, the sum of their execution time is at most 2^k. Formally, we require

$$\sum_{\tau \in T : d_\tau \in (2^{k-1}, 2^k]} c_{i,\tau} y_{i,\tau} \leq 2^k \qquad \forall i \in M,\ \forall k \in \mathbb{N}. \tag{9}$$

We call these conditions the *relaxed dbf constraints*. It is clear that these constraints have to be fulfilled by any feasible task assignment. Since they are linear, they can be used in an LP relaxation for the problem. Their sparsity gives the potential to use the rounding procedure from Sect. 3 to obtain an integral solution that violates the relaxed dbf constraints only by a constant factor. The following lemma shows that—even when violated up to a constant factor—the presented constraints are approximately sufficient.

Lemma 2 *Let $\mathcal{T}$ be an assignment for the task system T such that, for all machines i, $\sum_{\tau \in \mathcal{T}_i} u_{i,\tau} \leq \beta$ and $\sum_{\tau \in \mathcal{T}_i : d_\tau \in (2^{k-1}, 2^k]} c_{i,\tau} \leq \beta\, 2^k$. Then $dbf_{\mathcal{T},i}(s) \leq 6\beta s$ for all $s \geq 0$ and $\mathcal{T}$ is a feasible task assignment under a speedup factor of 6β.*

Proof Let $\pi \in \mathbb{N}$ and $s := 2^\pi$. Consider an assignment $\mathcal{T}$ of the tasks in T to the machines in M. For any machine $i \in M$, we bound $dbf_i(s) := dbf_{\mathcal{T},i}(s)$ by

$$\begin{aligned}
dbf_i(s) &= \sum_{\tau \in \mathcal{T}_i : d_\tau \leq s} \left\lfloor \frac{s + t_\tau - d_\tau}{t_\tau} \right\rfloor c_{i,\tau} \\
&\leq \sum_{\tau \in \mathcal{T}_i : d_\tau \leq s} \left(s \frac{c_{i,\tau}}{t_\tau} + c_{i,\tau} \right) \\
&\leq s \sum_{\tau \in \mathcal{T}_i} \frac{c_{i,\tau}}{t_\tau} + \sum_{\tau \in \mathcal{T}_i : d_\tau \leq s} c_{i,\tau} \\
&= s \sum_{\tau \in \mathcal{T}_i} u_{i,\tau} + \sum_{k=0}^{\log_2(s)} \sum_{\tau \in \mathcal{T}_i : d_\tau \in (2^{k-1}, 2^k]} c_{i,\tau}
\end{aligned}$$

$$
\begin{aligned}
&\leq \beta s + \sum_{k=0}^{\pi} \beta\, 2^k \\
&\leq \beta s + \beta\, 2^{\pi+1} \\
&= \beta s + 2\beta s = 3\beta s.
\end{aligned}
$$

Hence, for any machine i and for arbitrary s we get that $dbf_i(s) \leq dbf_i(2^{\lceil \log_2 s \rceil}) \leq 3\beta\, 2^{\lceil \log_2 s \rceil} \leq 6\beta s$. This implies that $\mathcal{T}$ is a feasible assignment for scheduling the tasks in T to the set of unrelated machines whenever the machines receive a speedup factor of 6β. □

Let $\rho > 1$. We define the function $r(x) := \rho^{\lceil \log_\rho x \rceil}$. Assume we are given an instance of our problem. Let $d_{\max} := \max_{\tau \in T} d_\tau$ and define the set $\mathcal{D}_\rho := \{\rho^0, \rho^1, \ldots, r(d_{\max})\}$. We now formulate the following linear program, denoted by Ass-LP, where the deadlines are rounded up to the nearest power of ρ rather than to the nearest power of 2 (as implicitly done in the relaxed dbf constraints given above).

$$\sum_{i \in M} y_{i,\tau} = 1 \qquad \forall \tau \in T, \tag{10a}$$

$$\sum_{\tau \in T} u_{i,\tau}\, y_{i,\tau} \leq 1 \qquad \forall i \in M, \tag{10b}$$

$$\sum_{\tau \in T : r(d_\tau) = D} c_{i,\tau}\, y_{i,\tau} \leq D \qquad \forall D \in \mathcal{D}_\rho,\ \forall i \in M, \tag{10c}$$

$$y_{i,\tau} \geq 0 \qquad \forall \tau \in T, \forall i \in M : u_{i,\tau} \leq 1 \wedge c_{i,\tau} \leq d_\tau. \tag{10d}$$

If Ass-LP is infeasible, then there can be no feasible (integral) task assignment. Now assume that it is feasible and we have computed a feasible solution $\mathbf{y}^*$. For each machine i and deadline $D \in \mathcal{D}_\rho$ we extract a value

$$U_{i,D} := \sum_{\tau \in T : r(d_\tau) = D} c_{i,\tau}\, y^*_{i,\tau}.$$

Based on these values, we define a strengthened variation of Ass-LP, denoted by SAss-LP in the sequel. We obtain the latter by replacing the constraints (10c) by the following set of constraints:

$$\sum_{\tau \in T : r(d_\tau) = D} c_{i,\tau}\, y_{i,\tau} \leq U_{i,D} \qquad \forall D \in \mathcal{D}_\rho,\ \forall i \in M. \tag{10c$'$}$$

Clearly if $\mathbf{y}^*$ is a feasible solution for Ass-LP, it is also a feasible solution for SAss-LP and if SAss-LP is infeasible, then no feasible task assignment exists. We now round $\mathbf{y}^*$ to an integral solution which *approximately* satisfies SAss-LP by using the rounding procedure presented in Sect. 3. Note that each machine-task combination appears in one constraint of type (10b) and in one of type (10c′). Hence, when applying Theorem 1, we have that $\gamma = 2$. Therefore, applying the rounding procedure from

Theorem 1 gives a task assignment $\hat{\mathbf{y}}$ that satisfies constraints (10a) and (10d) and the following two inequalities

$$\sum_{\tau \in T} u_{i,\tau} \hat{y}_{i,\tau} \leq 3 \qquad \forall i \in M, \tag{11}$$

$$\sum_{\tau \in T : r(d_\tau) = D} c_{i,\tau} \hat{y}_{i,\tau} \leq U_{i,D} + 2D \qquad \forall D \in \mathcal{D}_\rho,\ \forall i \in M. \tag{12}$$

Observe that $U_{i,D} \leq D$ for all $D \in \mathcal{D}_\rho$ and all machines i, and hence the vector $\hat{\mathbf{y}}$ violates the relaxed dbf-constraints by at most a factor of $\beta = 3$. Hence, Lemma 2 directly implies that the task assignment given by the vector $\hat{\mathbf{y}}$ is feasible with a speedup of 18 if we choose $\rho = 2$. However, using the definition of $U_{i,D}$ and a more careful calculation, we can bound the needed speedup even further.

Lemma 3 *If we choose $\rho = 1 + \sqrt{6}/3$, the task assignment implied by the variables $\hat{\mathbf{y}}$ is feasible if the machines run at speed $8 + 2\sqrt{6}$.*

Proof Let $\mathcal{T}$ denote the assignment implied by the integral variables $\hat{\mathbf{y}}$. Correspondingly, $\mathcal{T}_i$ denotes the set of tasks that are assigned to machine i, that is $\mathcal{T}_i = \{\tau \in T : \hat{y}_{i,\tau} = 1\}$. We show that for the task assignment implied by $\hat{\mathbf{y}}$, $dbf_{\mathcal{T},i}(s) \leq (8+2\sqrt{6})s$ for all s and any machine i, if we choose $\rho = 1 + \sqrt{6}/3$:

$$\begin{aligned}
dbf_{\mathcal{T},i}(s) &= \sum_{\tau \in \mathcal{T}_i : d_\tau \leq s} \left\lfloor \frac{s + t_\tau - d_\tau}{t_\tau} \right\rfloor c_{i,\tau} \\
&\leq \sum_{\tau \in \mathcal{T}_i : r(d_\tau) \leq r(s)} \left(s \frac{c_{i,\tau}}{t_\tau} + c_{i,\tau} \right) \\
&\leq s \sum_{\tau \in T} \frac{c_{i,\tau}}{t_\tau} \hat{y}_{i,\tau} + \sum_{\tau \in T : r(d_\tau) \leq r(s)} c_{i,\tau}\, \hat{y}_{i,\tau} \\
&= s \sum_{\tau \in T} u_{i,\tau} \hat{y}_{i,\tau} + \sum_{k=0}^{\log_\rho(r(s))} \sum_{\tau \in T : r(d_\tau) = \rho^k} c_{i,\tau}\, \hat{y}_{i,\tau} \\
&\overset{(11)\&(12)}{\leq} 3s + \sum_{k=0}^{\log_\rho(r(s))} \left(U_{i,\rho^k} + 2\rho^k \right) \\
&\leq 3s + \rho^{\log_\rho(r(s))} + 2 \sum_{k=0}^{\log_\rho(r(s))} \rho^k \\
&= 3s + r(s) + 2\, \frac{\rho^{\log_\rho(r(s))+1} - 1}{\rho - 1} \\
&\leq 3s + r(s) + 2\rho \frac{r(s)}{\rho - 1} \\
&\leq s(3 + \rho + 2\frac{\rho^2}{\rho - 1}) = \left(8 + 2\sqrt{6}\right) s.
\end{aligned}$$

The last inequality follows since $r(s) \leq \rho s$, whereas the last equality follows by optimization of the value of ρ, i.e., we set $\rho := 1 + \sqrt{6}/3$. □

Theorem 2 *There is a* $(8 + 2\sqrt{6})$*-approximation test for the problem of assigning tasks to unrelated machines.*

Proof We solve the linear program Ass-LP and if feasible, we formulate the program SAss-LP given by (10a), (10b), (10c′) and (10d). We then apply Theorem 1 to obtain the integral vector $\hat{\mathbf{y}}$, noting that $\gamma = 2$. The result follows from choosing $\rho = 1 + \sqrt{6}/3$ and applying Lemma 3. □

4.2 Hardness result

Finally, we show that it is *NP*-hard to decide whether a task system T has an assignment which is feasible on m unrelated machines, even with a speedup factor of $2-\epsilon$, for any $\epsilon > 0$; the proof follows the lines of the $(\frac{3}{2} - \epsilon)$-hardness result for makespan minimization in [26]. We reduce from the 3- DIMENSIONAL MATCHING problem which is known to be *NP*-complete [26] and is defined below.

Instance of 3- DIMENSIONAL MATCHING Given are three disjoint sets, consisting of n elements each: $A = \{a_1, a_2, \ldots, a_n\}$, $B = \{b_1, b_2, \ldots, b_n\}$ and $C = \{c_1, c_2, \ldots, c_n\}$. Also, given is a family $F = \{F_1, F_2, \ldots, F_m\}$ of $m \geq n$ triples such that each triple contains exactly one element from the set A, one from B, and one from C, that is, $|F_i \cap A| = |F_i \cap B| = |F_i \cap C| = 1$ for all $i \in \{1, \ldots, m\}$.

Question Does F contain a 3-dimensional matching, i.e., a subfamily $F' \subseteq F$ for which $|F'| = n$ and $\bigcup_{F_i \in F'} F_i = A \cup B \cup C$?

Let $\epsilon > 0$. Consider an instance $\mathcal{I}$ of 3- DIMENSIONAL MATCHING. We create an instance (task system) T for sporadic real-time scheduling on unrelated parallel machines in the following way:

- Define a large constant $M := 2/\epsilon$.
- Associate a machine i with each triple F_i, yielding m machines.
- Let $F(a_k) \subseteq F$ be the set of triples containing element $a_k \in A$. We will slightly abuse notation and also use $F(a_k)$ to refer to the set of machines corresponding to the triples in $F(a_k)$. Further, let $f(a_k) := |F(a_k)|$. Note that $\sum_{a_k \in A} f(a_k) = m$.
- For each element $b_g \in B$, create one task τ_g such that $d_{\tau_g} = 1$ and $t_{\tau_g} = \infty$. Further, $c_{i,\tau_g} = 1$ if $b_g \in F_i$, and $c_{i,\tau_g} = 2$ otherwise. We refer to these tasks as being type I tasks. There will be n tasks of type I.
- For each element $c_l \in C$, create one task τ_l such that $d_{\tau_l} = M$ and $t_{\tau_l} = \infty$. Further, $c_{i,\tau_l} = M - 1$ if $c_l \in F_i$, and $c_{i,\tau_l} = 2M$ otherwise. We refer to these tasks as being type II tasks. There will be n tasks of type II.
- For each element $a_k \in A$, create $f(a_k) - 1$ dummy tasks $dum(k_1), \ldots dum(k_{f(a_k)-1})$ which have a deadline and period equal to one. Each dummy task $dum(k_z)$ has $c_{i,k_z} = 1$ if $i \in F(a_k)$, and $c_{i,k_z} = 2$ otherwise. Note that in total there will be $m - n$ dummy tasks.

In Lemmas 4 and 5 we show that the reduction from 3- DIMENSIONAL MATCHING is valid.

Lemma 4 *If there exists a 3-dimensional matching for $\mathcal{I}$, then there exists a feasible assignment for T.*

Proof Let $F^* \subseteq F$ be the triples in the 3-dimensional matching. For all triples $F_i = \{a_k, b_g, c_l\} \in F^*$, schedule tasks τ_g and τ_l on machine i. Note that machine i can process the tasks assigned to it. Also, all tasks corresponding to elements in B and C have been scheduled. As the element a_k is only covered by one triple in G^*, it follows that there are $f(a_k) - 1$ machines remaining in $F(a_k)$ which are not assigned any tasks yet. Assign the $f(a_k) - 1$ dummy tasks $dum(k_1), \ldots dum(k_{f(a_k)-1})$ to these machines. Note that a machine in $F(a_k)$ can process exactly one dummy task corresponding to the element a_k. □

Before giving Lemma 5, we first need to show some propositions about the elements in the created scheduling instance T.

Proposition 2 *For any $\rho < 2$, no task τ_g corresponding to an element $b_g \in B$ can be scheduled on a machine i if $b_g \notin F_i$, even under a speedup of ρ.*

Proposition 3 *For any $\rho < 2$, no task τ_l corresponding to an element $c_l \in C$ can be scheduled on a machine i if $c_l \notin F_i$, even under a speedup of ρ.*

Proposition 4 *For any $\rho < 2$, no dummy task $dum(k_z)$ corresponding to an element $a_k \in A$ can be scheduled on a machine i if $i \notin F(a_k)$, even under a speedup of ρ.*

Proof We argue for Proposition 2. Propositions 3 and 4 follow similarly. The proof is by contradiction; let a task τ_g corresponding to an element $b_g \in B$ be scheduled on a machine i such that $b_g \notin F_i$. Then $c_{i,\tau_h} = 2$. At time $d_{\tau_g} = 1$ the first job of task τ_g needs to be completed and hence a speedup of 2 is required. □

Proposition 5 *For any $\rho < 2 - \epsilon/2$, no dummy task $dum(k_z)$ can be scheduled together with another task on the same machine, even under a speedup of ρ.*

Proof We argue by contradiction. We consider three cases:

- Suppose that two dummy tasks were scheduled on the same machine. Both dummies would need to finish their first job by their first deadline which is at time 1. Their accumulated processing requirement to the machine would be at least 2 and hence a speedup of at least 2 would be required.
- Suppose a dummy task and a task of type I were scheduled on the same machine. We reason analogously to the previous case.
- Suppose a dummy task and a task of type II were scheduled on the same machine. Consider time instant M, where the dummy task should have finished M jobs, whereas the other task should have finished its first job. The accumulated processing requirement for the machine by time M would be at least $M * 1 + (M - 1) = 2M - 1$. Hence, a speedup of $\frac{2M-1}{M} = 2 - \frac{1}{M} = 2 - \epsilon/2 > \rho$ would be required, by our definition of M. □

Lemma 5 *For any $\rho < 2 - \epsilon/2$, if there exists a feasible assignment for T with speedup ρ, then there exists a 3-dimensional matching for $\mathcal{I}$.*

Proof Proposition 5 yields that each dummy task gets its own machine, even under a speedup of $\rho < 2-\epsilon/2$. Therefore, and by Proposition 4, for all $a_k \in A$, there remains in each group $F(a_k)$ one machine available to process tasks of type I or II, even under a speedup of $\rho < 2$. In total there are $m - (m - n) = n$ machines left which do not process a dummy task. There are $2n$ tasks of type I and II. Since a single machine cannot process two tasks of the same type under a speedup less than 2, it follows that each machine which does not process a dummy task, processes one task of type I and one task of type II. Let $i_k \in F(a_k)$ be the machine which does not process a dummy task but instead one task of type I and one task of type II. By Propositions 2 and 3, the only way for machine i_k to be feasible, even under a speedup of $\rho < 2$, is when it processes the tasks corresponding to b_g and c_l where $F_{i_k} = \{a_k, b_g, c_l\}$. It follows that the machines i_k, for $k \in \{1, \ldots, m\}$, define a 3-dimensional matching for $\mathcal{I}$. □

Theorem 3 *Let $\epsilon > 0$. There is no $(2 - \epsilon)$-approximation test for the problem of assigning tasks to unrelated machines, unless $P = NP$.*

Proof Theorem 3 follows by Lemmas 4 and 5, and by the 3- DIMENSIONAL MATCHING problem being *NP*-complete. □

We remark that our hardness result is different from the one by Andersson and Tovar [3]. They show that any algorithm needs a speedup factor of at least $2 - \epsilon$ for finding a feasible partition on m related parallel machines for a given task system with implicit deadlines in case the task system is feasible when migration is allowed.

5 Constant number of machines

Assuming that the number of machines m is bounded by a constant, in this section we present a polynomial-time dynamic programming (DP) algorithm that gives a $(1+\epsilon)$-approximation test for any $\epsilon > 0$. During phase p, $p = 1, 2, \ldots, n$, the DP computes a possible assignment of task τ_p, using assignments of the first $(p-1)$ tasks. In order to obtain a DP-table of bounded size we introduce an approximation of the demand bound function such that the contribution of each task can be derived by using only a constant number of values.

By scaling all parameters, we assume that $d_{\min} = \min_{\tau \in T} d_\tau = 1$. Let $\epsilon > 0$, and assume without loss of generality that $\epsilon < 1/2$. Let L be the minimum integer which satisfies $1 \leq (1+\epsilon)^{L-1}\epsilon^2$. We define the function dbf^* as

$$dbf_i^*(\tau, s) := \begin{cases} \left\lfloor \frac{s+t_\tau-d_\tau}{t_\tau} \right\rfloor c_{i,\tau} & \text{if } s < (1+\epsilon)^L d_\tau, \\ \frac{c_{i,\tau}}{t_\tau} s & \text{otherwise.} \end{cases}$$

Given a task assignment $\mathcal{T}$ of the tasks in T to the machines, we define

$$dbf_{\mathcal{T},i}^*(s) := \sum_{\tau \in \mathcal{T}_i} dbf_i^*(\tau, s) \quad \forall s > 0.$$

Further, to have clean notation, we write $dbf_i^*(s)$ instead of $dbf_{\mathcal{T},i}^*(s)$ in case the assignment $\mathcal{T}$ is clear from the context. The key observation is that for computing the function $dbf_i^*(\tau, s)$ for a fixed task τ, it suffices to know the utilization of the task τ and the values of the demand bound function $dbf_i(\tau, s)$ for $s \in \left[d_\tau, (1+\epsilon)^L d_\tau\right)$. Exploiting the properties of the functions $dbf_{\mathcal{T},i}(s)$ and $dbf_{\mathcal{T},i}^*(s)$ yields that $dbf_{\mathcal{T},i}^*$ is a $(1+\epsilon)$-approximation of the 'real' demand bound function $dbf_{\mathcal{T},i}$.

Lemma 6 *Given an assignment $\mathcal{T}$ and a constant $\epsilon < 1/2$. Then, for all machines i,*

(i) if $dbf_{\mathcal{T},i}^*((1+\epsilon)^k) \leq \alpha(1+\epsilon)^k$ for all $k \in \mathbb{N}_{\geq 0}$, then $dbf_{\mathcal{T},i}(s) \leq (1+\epsilon)^2 \alpha s$ for all $s \geq 0$;
(ii) if $dbf_{\mathcal{T},i}(r) \leq r$ for all $r \geq 0$, then $dbf_{\mathcal{T},i}^*(s) \leq (1+\epsilon)s$ for all $s \geq 0$.

Proof For the first claim, we first show that a slightly stronger statement holds for all powers of $1+\epsilon$. We show that if for all $k \in \mathbb{N}_{\geq 0}$ it holds that $dbf_{\mathcal{T},i}^*((1+\epsilon)^k) \leq \alpha(1+\epsilon)^k$, then for all $k \in \mathbb{N}_{\geq 0}$ it holds that

$$dbf_{\mathcal{T},i}\left((1+\epsilon)^k\right) \leq (1+\epsilon)\alpha(1+\epsilon)^k. \tag{13}$$

Subsequently, we show that then for all s it holds that $dbf_{\mathcal{T},i}(s) \leq (1+\epsilon)^2 \alpha s$.

Inequality (13) trivially holds for all k with $(1+\epsilon)^k \leq d_{\min}$, as then $dbf_i(\tau, (1+\epsilon)^k) = 0$ for all tasks $\tau \in T$. Assume that (13) holds for all $k' \in \mathbb{N}_{\geq 0}$ with $k' < k$, we show that it then also holds for k. Let $s := (1+\epsilon)^k$. Consider some partition of the tasks $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_m\}$. Let $\mathcal{T}_i^{\text{early}} := \{\tau \in \mathcal{T}_i | (1+\epsilon)^L d_\tau < s\}$ and $\mathcal{T}_i^{\text{late}} := \mathcal{T}_i \backslash \mathcal{T}_i^{\text{early}}$. By definition of $dbf_{\mathcal{T},i}^*$ it follows that

$$dbf_{\mathcal{T},i}(s) \leq \sum_{\tau \in \mathcal{T}_i} dbf_i^*(\tau, s) + \sum_{\tau \in \mathcal{T}_i^{\text{early}}} c_{i,\tau} = dbf_{\mathcal{T},i}^*(s) + \sum_{\tau \in \mathcal{T}_i^{\text{early}}} c_{i,\tau}. \tag{14}$$

Further, since for $\tau \in \mathcal{T}_i^{\text{early}}$ it holds that $d_\tau < \frac{s}{(1+\epsilon)^L}$,

$$\begin{aligned}
\sum_{\tau \in \mathcal{T}_i^{\text{early}}} c_{i,\tau} &\leq \sum_{\tau \in \mathcal{T}_i^{\text{early}}} \left\lfloor \frac{\frac{s}{(1+\epsilon)^L} + t_\tau - d_\tau}{t_\tau} \right\rfloor c_{i,\tau} \\
&= \sum_{\tau \in \mathcal{T}_i^{\text{early}}} dbf_i\left(\tau, \frac{s}{(1+\epsilon)^L}\right) \leq dbf_{\mathcal{T},i}\left(\frac{s}{(1+\epsilon)^L}\right) \\
&\overset{(*)}{\leq} \frac{\alpha s}{(1+\epsilon)^{L-1}} \leq \epsilon^2 \alpha s \leq \epsilon \alpha s.
\end{aligned} \tag{15}$$

The inequality at $(*)$ is due to the induction hypothesis; we assumed Inequality (13) to be correct for all $k' < k$ and $\left(\frac{s}{(1+\epsilon)^L}\right) = (1+\epsilon)^{k'}$ for $k' = k - L$. (Note that although this was only shown for $k' \in \mathbb{N}_{\geq 0}$, it holds also if $k' = k - L < 0$, since then $(1+\epsilon)^{k'} \leq 1 = d_{\min}$.) The penultimate inequality follows from $\frac{1}{(1+\epsilon)^{L-1}} < \epsilon^2$.

Inequalities (14) and (15) imply that $dbf_{\mathcal{T},i}(s) \leq dbf^*_{\mathcal{T},i}(s) + \epsilon\alpha s \leq (1+\epsilon)\alpha s$. For all values of s which are *not* powers of $1+\epsilon$, we observe that the function $dbf_{\mathcal{T},i}(s)$ is non-decreasing and thus if Inequality (13) holds for all $k \in \mathbb{N}_{\geq 0}$, then (i) holds for all s. That is, for values s that are not powers of $1+\epsilon$ we need an additional factor of $1+\epsilon$.

For the second claim no induction is necessary and we calculate that

$$\begin{aligned}
dbf^*_{\mathcal{T},i}(s) &< \sum_{\tau \in \mathcal{T}_i^{\text{late}}} dbf_i(\tau, s) + \sum_{\tau \in \mathcal{T}_i^{\text{early}}} \frac{s - d_\tau + \frac{s}{(1+\epsilon)^L}}{t_\tau} c_{i,\tau} \\
&\leq \sum_{\tau \in \mathcal{T}_i^{\text{late}}} dbf_i(\tau, s) + \sum_{\tau \in \mathcal{T}_i^{\text{early}}} \left(\left\lfloor \frac{s - d_\tau + t_\tau}{t_\tau} \right\rfloor + \frac{s}{(1+\epsilon)^L t_\tau} \right) c_{i,\tau} \\
&\leq dbf_{\mathcal{T},i}(s) + \sum_{\tau \in \mathcal{T}_i^{\text{early}}} (\epsilon^2 u_{i,\tau}) s \\
&= dbf_{\mathcal{T},i}(s) + \epsilon^2 dbf^*_{\mathcal{T}_i^{\text{early}},i}(s) \\
&\leq dbf_{\mathcal{T},i}(s) + \epsilon^2 dbf^*_{\mathcal{T},i}(s).
\end{aligned}$$

Therefore, $(1-\epsilon^2) dbf^*_{\mathcal{T},i}(s) \leq dbf_{\mathcal{T},i}(s)$, i.e., $dbf^*_{\mathcal{T},i}(s) \leq (1+\epsilon) dbf_{\mathcal{T},i}(s)$, for any $\epsilon < 1/2$. □

Note that, in contrast to other approximations of the demand bound function considered in the literature (e.g., [1]), in Lemma 6 we do not use an analysis task by task, and we do not bound the ratio $dbf(\tau, s)/dbf^*(\tau, s)$. In fact, the latter can be unbounded: consider for example a task τ with $c_\tau = 1$, $d_\tau = 1$, and $p_\tau = M$ for a very large value M, then $dbf(\tau, s) \geq 1$ for all $s \geq 1$ whereas $dbf^*(\tau, (1+\epsilon)^L) = (1+\epsilon)^L / M$.

Observe that Lemma 6 implies that at the cost of a $(1+\epsilon)^2$-speedup it suffices to check whether the condition $dbf^*_{\mathcal{T},i}(s) \leq s$ is (approximately) satisfied at powers of $1+\epsilon$. Therefore, the DP may characterize each task τ only by its utilization and the constantly many values $dbf^*_i(\tau, (1+\epsilon)^k)$ (namely those values for integers k such that $d_\tau \leq (1+\epsilon)^k < (1+\epsilon)^L d_\tau$, for each machine i).

For each task τ, each machine i and all $\ell \in \mathbb{N}_{\geq 0}$, we introduce a vector $v(i, \tau)$ by defining position

$$v(i, \tau)_\ell := \frac{dbf^*_i(\tau, (1+\epsilon)^\ell)}{(1+\epsilon)^\ell}.$$

Recall that for any vector $\mathbf{a}$ the infinity norm $\|\mathbf{a}\|_\infty = \max_i \{a_i\}$. The following proposition follows by definition.

Proposition 6 *Consider an assignment $\mathcal{T}$. For all machines $i \in M$, it holds that $\left\| \sum_{\tau \in \mathcal{T}_i} v(i, \tau) \right\|_\infty \leq \alpha$ if and only if $dbf^*_{\mathcal{T},i}((1+\epsilon)^\ell) \leq \alpha (1+\epsilon)^\ell$, for all $\ell \in \mathbb{N}_{\geq 0}$.*

We present a dynamic programming algorithm which either (1) asserts that there is no feasible assignment of the tasks to the machines by showing that there is no

assignment $\mathcal{T}$ of tasks to machines such that we have $\left\|\sum_{\tau\in\mathcal{T}_i} v(i,\tau)\right\|_\infty \leq 1+\epsilon$ for each machine i, or (2) finds an assignment $\mathcal{T}$ such that $\left\|\sum_{\tau\in\mathcal{T}_i} v(i,\tau)\right\|_\infty \leq 1+O(\epsilon)$ for each machine i. In the latter case, Lemma 6 and the above proposition imply an approximation test for the problem of assigning tasks to a constant number of unrelated machines. The test either concludes that the task system is not feasible (without speedup) or provides an assignment which is feasible in case the machines have a speedup factor of $1+O(\epsilon)$.

Assume without loss of generality that the tasks $\tau_1,\ldots,\tau_n$ are ordered such that $d_{\tau_p} \leq d_{\tau_{p+1}}$ for each p. We partition the tasks into groups $G_k := \{\tau | (1+\epsilon)^k \leq d_\tau < (1+\epsilon)^{k+1}\}$ for each $k \in \mathbb{N}_{\geq 0}$. The proposed DP works in phases; one phase for each task. The key idea is that when trying to assign task $\tau \in G_k$, the DP needs only a constant number of values from the assignment of the previously considered tasks.

Define $L^{(k)} := \min\{k, L\}$ (such that $k - L^{(k)} \geq 0$). For all tasks having a deadline at most a factor $(1+\epsilon)^L$ smaller than d_{τ_p}, the DP needs to know how much the vectors of tasks from each group $G_{k'}$ (with $k - L^{(k)} < k' \leq k$) contribute towards dimension ℓ on machine i, for $k \leq \ell \leq k+L$. For the same groups $G_{k'}$ the DP needs to know the summed utilization per machine i over group $G_{k'}$. For the other groups $G_{k'}$ with $k' \leq k - L^{(k)}$ for each machine i, only the aggregated utilization over all groups is needed. Formally we need, for all i,

- the sum $\sum_{\tau\in\mathcal{T}_i\cap\left(\bigcup_{k'=0}^{k-L^{(k)}} G_{k'}\right)} u_{i,\tau}$,
- the sum $\sum_{\tau\in\mathcal{T}_i\cap G_{k'}} u_{i,\tau}$, for all k' s.t. $k - L^{(k)} < k' \leq k$, and
- the sum $\sum_{\tau\in\mathcal{T}_i\cap G_{k'}} v(i,\tau)_\ell$, for all ℓ s.t. $k \leq \ell \leq k+L$ and all k' s.t. $\ell - L^{(\ell)} < k' \leq k$.

Ideally, we would like the DP to store all possible combinations of the above quantities that can result from assigning the tasks of previous iterations. Then, the DP could compute the values for the next iteration by taking each combination of values from the last iteration and additionally schedule task τ to one of the machines. Unfortunately, the number of possible combinations of the above values is not polynomially bounded, as already the input values (which then imply the utilization, etc.) might be in an exponential range. In order to bound them, we round entries of the vectors $v(i,\tau)$. The DP then performs the described procedure with the rounded vectors. This will result in a polynomial-time procedure.

Consider a task $\tau \in G_k$, $k \in \mathbb{N}_{\geq 0}$. For all i and τ, define $v'(i,\tau)_\ell := \frac{\epsilon}{n}\left\lfloor \frac{n}{\epsilon} v(i,\tau)_\ell \right\rfloor$ for each $\ell \leq k+L$, and $v'(i,\tau)_\ell := u'_{i,\tau} := \frac{\epsilon}{n}\left\lfloor \frac{n}{\epsilon} u_{i,\tau} \right\rfloor$ for each $\ell > k+L$. The following lemma bounds the rounding error.

Lemma 7 *Let i be a machine and $\mathcal{T}_i$ be a set of tasks. For all $\ell \in \mathbb{N}_{\geq 0}$, it holds that $\sum_{\tau\in\mathcal{T}_i} v'(i,\tau)_\ell \leq \sum_{\tau\in\mathcal{T}_i} v(i,\tau)_\ell \leq \sum_{\tau\in\mathcal{T}_i} v'(i,\tau)_\ell + \epsilon$.*

Proof Consider a task $\tau \in \mathcal{T}_i$. Define $k(\tau)$ such that $\tau \in G_{k(\tau)}$. We show for each $\tau \in \mathcal{T}_i$ and the corresponding $k(\tau)$ that $v'(i,\tau)_\ell \leq v(i,\tau)_\ell \leq v'(i,\tau)_\ell + \epsilon/n$. The statement trivially follows. The case where $\ell \leq k(\tau)+L$ follows trivially from the relation $\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$ which holds for any x. Therefore, consider the

case $\ell > k(\tau) + L$. Since $\tau \in G_{k(\tau)}$, it follows that $d_\tau < (1+\epsilon)^{k(\tau)+1}$. Hence, $dbf_i^*(\tau, (1+\epsilon)^\ell) = \frac{c_{i,\tau}}{t_\tau}(1+\epsilon)^\ell = u_{i,\tau}(1+\epsilon)^\ell$ which yields $v(i,\tau)_\ell = u_{i,\tau}$. The result for the case $\ell > k(\tau) + L$ now also follows from $\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$. □

Note that each rounded vector $v'(i,\tau)$ can be described with only constantly many pieces of information. When working with the rounded vectors, for the quantities mentioned above, there are only a polynomial number of combinations (assuming that m is a constant). In particular, the dynamic programming table will be of polynomial size. Formally, the dynamic programming table consists of entries of the form $(p, \mathbf{z}, \mathbf{w}, \mathbf{c})$ where

- $p \in \{0, \ldots, n\}$ denotes the phase of the DP. In phase p, task τ_p is being assigned to a machine. Let k be an integer such that $\tau_p \in G_k$;
- in phase p, for each machine i, the value $z_i^{(p)}$ is of the form $\ell \frac{\epsilon}{n}$ for some integer ℓ, denoting the rounded aggregated utilization for machine i due to the tasks having a deadline at least a factor of $(1+\epsilon)^L$ smaller with respect to the deadline of task τ_p;
- in phase p, for each machine i and each k' with $k - L^{(k)} < k' \leq k$, the value $w_{i,k'}^{(p)}$ is of the form $\ell \frac{\epsilon}{n}$ for some integer ℓ, denoting the rounded utilization of tasks in $G_{k'} \cap \mathcal{T}_i$;
- in phase p, for each triple $(i, k', k'') \in C_p$ with $C_p = \{(i, k', k'') : 1 \leq i \leq m;\ k \leq k'' < k + L$ and $k'' - L^{(k'')} < k' \leq k\}$, the value $c_{i,k',k''}^{(p)}$ is of the form $\ell \frac{\epsilon}{n}$ for some integer ℓ, denoting the quantity $\sum_{\tau \in \mathcal{T}_i \cap G_{k'}} v'(i,\tau)_{k''}$, expressing how much the vectors of the tasks in $G_{k'}$ on machine i contribute towards dimension k''.

We require the following set of conditions to be satisfied for a DP-cell $(p, \mathbf{z}, \mathbf{w}, \mathbf{c})$ to exist; for each machine $i \in M$ and all $k'' \in \{k, \ldots, k+L\}$

$$z_i + \sum_{k'=k-L^{(k)}+1}^{k''-L^{(k'')}} w_{i,k'} + \sum_{k'=k''-L^{(k'')}+1}^{k} c_{i,k',k''} \leq 1 + \epsilon. \tag{16}$$

This condition implies that, for all parameters, $z_i,\ w_{i,k'},\ c_{i,k',k''} \leq 1 + \epsilon$.

Proposition 7 *The number of DP-cells is bounded by* $n((1+\epsilon)n/\epsilon)^{2mL^2}$.

Proof The values $z_i,\ w_{i,k'}$ and $c_{i,k',k''}$ are all stored with accuracy $\frac{\epsilon}{n}$ and hence each of those can take $(1+\epsilon)n/\epsilon$ many different realizations. Further, i can take m different values whereas k' and k'' can take L different values if the DP is in a certain phase p. Further, there are at most n phases for the DP. It follows that the number of cells of the DP table is no more than

$$n \cdot \left(\frac{(1+\epsilon)n}{\epsilon}\right)^m \cdot \left(\frac{(1+\epsilon)n}{\epsilon}\right)^{mL} \cdot \left(\frac{(1+\epsilon)n}{\epsilon}\right)^{mL^2} \leq n\left(\frac{(1+\epsilon)n}{\epsilon}\right)^{2mL^2}$$

□

Each entry $(p, \mathbf{z}, \mathbf{w}, \mathbf{c})$ of the DP-table either stores 'YES' or 'NO' depending on whether or not there is an assignment of the tasks $\tau_1, \ldots, \tau_p$ to the machines which yields the quantities given by the vectors $\mathbf{z}$, $\mathbf{w}$, $\mathbf{c}$.

We proceed by describing how to fill the DP-table. First, initialize the table by assigning a 'YES'-entry to $(0, \mathbf{0}, \mathbf{0}, \mathbf{0})$ and a 'NO'-entry to any other entry with $p = 0$. Assume that for some p, all entries of the form $(p-1, \mathbf{z}^{(p-1)}, \mathbf{w}^{(p-1)}, \mathbf{c}^{(p-1)})$ have been computed. The DP-table is then iteratively extended to phase p. Phase p considers each combination of assigning task τ_p to some machine i and a DP-cell $(p-1, \mathbf{z}^{(p-1)}, \mathbf{w}^{(p-1)}, \mathbf{c}^{(p-1)})$ with a 'YES'-entry. Intuitively, the DP computes which values for $\mathbf{z}^{(p)}, \mathbf{w}^{(p)}$, and $\mathbf{c}^{(p)}$ are obtained if it takes the task assignment encoded in the DP-cell $(p-1, \mathbf{z}^{(p-1)}, \mathbf{w}^{(p-1)}, \mathbf{c}^{(p-1)})$ and additionally schedules task τ_p to machine i.

Let tasks τ_{p-1} and τ_p be in group G_h and G_k, respectively. Almost all entries of the vectors are equal and hence we only list the values which differ with respect to the entries from phase $p-1$. If $h = k$ and task τ_p is assigned to machine i, then $w_{i,k}^{(p)} = w_{i,k}^{(p-1)} + u'_{i,\tau_p}$, and $c_{i,k,k''}^{(p)} = c_{i,k,k''}^{(p-1)} + v'(i, \tau_p)_{k''}$ for all $k'' \in \{k, \ldots, k+L\}$.

If $h \neq k$, w.l.o.g. we assume that $h = k-1$ (e.g., by creating dummy tasks of zero processing requirement to fill in-between groups that otherwise would be empty and thus, non-existent). Then, $z_g^{(p)} = z_g^{(p-1)} + w_{g,k-L^{(k)}}^{(p-1)}$ for all machines $g \in M$; as we have moved up one group, tasks from group $G_{k-L^{(k)}}$ now have their deadline at least a factor $(1+\epsilon)^L$ away from d_{τ_p} and their utilizations are not stored separately anymore, but in a aggregated way. For the tasks in groups $G_{k'}$ such that $k - L^{(k)} < k' < k$ and all machines $g \in M$ we still store the utilizations per group and thus $w_{g,k'}^{(p)} = w_{g,k'}^{(p-1)}$. For group G_k we also store the utilization for the machine i that it was assigned to, so $w_{i,k}^{(p)} = u'_{i,\tau_p}$ and $w_{g,k}^{(p)} = 0$ for all machines $g \neq i$. Since τ_p is assigned to machine i, $c_{i,k,k''}^{(p)} = v'(i, \tau_p)_{k''}$ for all $k'' : k \leq k'' \leq k+L$; and for all machines $g \neq i$, $c_{g,k,k''}^{(p)} = 0$ for all $k'' : k \leq k'' \leq k+L$. Finally, $c_{g,k',k''}^{(p)} = c_{g,k',k''}^{(p-1)}$ for all machines $g \in M$, all $k'' : k \leq k'' \leq k+L$ and all $k' : k'' - L^{(k'')} < k' < k$.

Hereafter, the DP checks whether the computed values $\mathbf{z}^{(p)}, \mathbf{w}^{(p)}$ and $\mathbf{c}^{(p)}$ satisfy the condition given in (16). If this is the case, then the corresponding DP-cell $(p, \mathbf{z}^{(p)}, \mathbf{w}^{(p)}, \mathbf{c}^{(p)})$ is filled with a 'YES'-entry and we say that this DP-cell *extends* the DP-cell $(p-1, \mathbf{z}^{(p-1)}, \mathbf{w}^{(p-1)}, \mathbf{c}^{(p-1)})$. In case there does not exist a DP-cell $(p-1, \mathbf{z}^{(p-1)}, \mathbf{w}^{(p-1)}, \mathbf{c}^{(p-1)})$ which can be extended to the DP-cell $(p, \mathbf{z}^{(p)}, \mathbf{w}^{(p)}, \mathbf{c}^{(p)})$, the latter DP-cell is filled with a 'NO'-entry.

The DP-table is filled inductively, phase by phase, until each cell in the DP-table is filled. In the next lemma we show for any machine i the equivalence between the inequalities $\left\| \sum_{\tau \in \mathcal{T}_i} v'(i, \tau) \right\|_\infty \leq 1 + \epsilon$ on the one hand, and condition (16) on the other hand.

Lemma 8 *For phase p, if there exists a DP-cell of the form $(p, \mathbf{z}^{(p)}, \mathbf{w}^{(p)}, \mathbf{c}^{(p)})$ with a 'YES'-entry, then there exists a task assignment $\mathcal{T}$ of the first p tasks to the machines, such that for each $i \in M$ it holds that $\left\| \sum_{\tau \in \mathcal{T}_i} v'(i, \tau) \right\|_\infty \leq 1 + \epsilon$.*

Proof Consider some phase p and assume that the statement is correct for all phases $p' < p$. Let k be such that $\tau_p \in G_k$. As no first job of any task in G_k has its deadline before $(1+\epsilon)^k$ it follows by induction that for all $\ell : 0 \leq \ell < k$ it holds that $\sum_{\tau \in \mathcal{T}_i} v'(i,\tau)_\ell \leq 1+\epsilon$. Next, we show the statement for phase p and the corresponding dimension k. Consider DP-cell of the form $(p, \mathbf{z}^{(p)}, \mathbf{w}^{(p)}, \mathbf{c}^{(p)})$ with a 'YES'-entry. We denote by $(16)^{(p)}$ the corresponding inequality given in (16) at phase p of the DP, that is, when task τ_p is being assigned to a machine. For all machines i it follows that,

$$\begin{aligned}\sum_{\tau \in \mathcal{T}_i} v'(i,\tau)_k &= \sum_{k'=0}^{k-L^{(k)}} \sum_{\tau \in \mathcal{T}_i \cap G_{k'}} u'_{i,\tau} + \sum_{k'=k-L^{(k)}+1}^{k} \sum_{\tau \in \mathcal{T}_i \cap G_{k'}} v'(i,\tau)_k \\ &= z_i^{(p)} + \sum_{k'=k-L^{(k)}+1}^{k} c_{i,k',k}^{(p)} \overset{(16)^{(p)}}{\leq} 1+\epsilon.\end{aligned}$$

The last inequality follows by setting the parameter k'' of $(16)^{(p)}$ equal to k. Next, consider the dimensions $\ell : k < \ell \leq k+L$. Note that $d_{\tau_p} < (1+\epsilon)^{k+1}$. Consequently, every task in $\cup_{k'=0}^{k} G_{k'}$ has its first deadline before $(1+\epsilon)^{k+1}$, that is, being in phase p of the DP where $\tau_p \in G_k$ it follows that $G_{k'} = \emptyset$ for all $k' > k$. This insight is used in the first equality below.

$$\begin{aligned}\sum_{\tau \in \mathcal{T}_i} v'(i,\tau)_\ell &= \sum_{k'=0}^{\ell-L^{(\ell)}} \sum_{\tau \in \mathcal{T}_i \cap G_{k'}} u'_{i,\tau} + \sum_{k'=\ell-L^{(\ell)}+1}^{k} \sum_{\tau \in \mathcal{T}_i \cap G_{k'}} v'(i,\tau)_\ell \\ &= \left(z_i^{(p)} + \sum_{k'=k-L^{(k)}+1}^{\ell-L^{(\ell)}} w_{i,k'}^{(p)} \right) + \sum_{k'=\ell-L^{(\ell)}+1}^{k} c_{i,k',\ell}^{(p)} \overset{(16)^{(p)}}{\leq} 1+\epsilon.\end{aligned}$$

The last inequality follows by setting the parameter k'' of $(16)^{(p)}$ equal to ℓ. Finally, the analysis for any dimension $\ell > k+L$ is equal to that of the dimension $\ell = k+L$ as the contribution of each task to any dimension $\ell \geq k+L$ will be approximated by its utilization. Thus, $\left\| \sum_{\tau \in \mathcal{T}_i} v'(i,\tau) \right\|_\infty \leq 1+\epsilon$ and the statement follows. □

Lemma 9 *For phase p, there exists a DP-cell of the form $(p, \mathbf{z}^{(p)}, \mathbf{w}^{(p)}, \mathbf{c}^{(p)})$ with a 'YES'-entry, if there exists a task assignment $\mathcal{T}$ of the first p tasks to the machines, such that for each $i \in M$ it holds that $\left\| \sum_{\tau \in \mathcal{T}_i} v'(i,\tau) \right\|_\infty \leq 1+\epsilon$.*

Proof Consider the assignment $\mathcal{T}^{(p)}$ where the first p tasks are assigned to the machines such that $\left\| \sum_{\tau \in \mathcal{T}_i^{(p)}} v'(i,\tau) \right\|_\infty \leq 1+\epsilon$ for all machines $i \in M$. Let $\tau_p \in G_k$. Define the following values for each machine i:

$$
\begin{aligned}
z_i^{(p)} &:= \sum_{k'=0}^{k-L^{(k)}} \sum_{\tau \in \mathcal{T}_i^{(p)} \cap G_{k'}} u'_{i,\tau}, \\
w_{i,k'}^{(p)} &:= \sum_{\tau \in \mathcal{T}_i^{(p)} \cap G_{k'}} u'_{i,\tau} && \text{for } k' \in \{k - L^{(k)}, \ldots, k\}, \text{ and} \\
c_{i,k',k''}^{(p)} &:= \sum_{\tau \in \mathcal{T}_i^{(p)} \cap G_{k'}} v'(i,\tau)_{k''} && \text{for } k'' \in \{k, \ldots, k + L\}, \\
& && \text{and for } k' \in \{k'' - L^{(k'')}, \ldots, k\}.
\end{aligned}
$$

First we need to show that the DP-cell $(p, \mathbf{z}^{(p)}, \mathbf{w}^{(p)}, \mathbf{c}^{(p)})$ exists, that is, we need to check whether the inequality $(16)^{(p)}$ holds. From the lemma statement it is known that $\sum_{\tau \in \mathcal{T}_i} v'(i,\tau)_\ell \leq 1 + \epsilon$, for all ℓ. Thus, in particular, this inequality also holds for all dimensions $k'' \in \{k, \ldots, k + L\}$. Therefore, for each machine i and each dimension $k'' \in \{k, \ldots, k + L\}$ it holds that

$$
\begin{aligned}
& z_i^{(p)} + \sum_{k'=k-L^{(k)}+1}^{k''-L^{(k'')}} w_{i,k'}^{(p)} + \sum_{k'=k''-L^{(k'')}+1}^{k} c_{i,k',k''}^{(p)} \\
&= \sum_{k'=0}^{k''-L^{(k'')}} \sum_{\tau \in G_{k'}} v'(i,\tau)_{k''} + \sum_{k'=k''-L^{(k'')}+1}^{k} c_{i,k',k''}^{(p)} \\
&= \sum_{k'=0}^{k''-L^{(k'')}} \sum_{\tau \in G_{k'}} v'(i,\tau)_{k''} + \sum_{k'=k''-L^{(k'')}+1}^{k} \sum_{\tau \in G_{k'}} v'(i,\tau)_{k''} \\
&= \sum_{\tau \in \mathcal{T}_i^{(p)}} v'(i,\tau)_{k''} \leq 1 + \epsilon.
\end{aligned}
$$

Consequently, the DP-cell $(p, \mathbf{z}^{(p)}, \mathbf{w}^{(p)}, \mathbf{c}^{(p)})$ exists. The DP-cell $(p, \mathbf{z}^{(p)}, \mathbf{w}^{(p)}, \mathbf{c}^{(p)})$ trivially extends the DP-cell $(p-1, \mathbf{z}^{(p-1)}, \mathbf{w}^{(p-1)}, \mathbf{c}^{(p-1)})$ by assigning task τ_p to machine i if $\tau_p \in \mathcal{T}_i^{(p)}$. By induction it follows that the DP-cell $(p, \mathbf{z}^{(p)}, \mathbf{w}^{(p)}, \mathbf{c}^{(p)})$ contains a 'YES'-entry, for all $p \in \{0, 1, \ldots, n\}$. □

Combining Lemmas 6, 7, 8 and 9, and Proposition 6 yields a $(1+7\epsilon)$-approximation test, for any ϵ and a constant number of machines. The claim on the running time follows from Proposition 7. If for a given $\epsilon > 0$ we run the above procedure with $\epsilon' := \epsilon/7$, rather than with ϵ, we obtain a $(1+\epsilon)$-approximation test.

Theorem 4 *For a constant number of machines and for any $\epsilon > 0$ there exists a $(1+\epsilon)$-approximation test, that runs in time polynomial in the number of tasks.*

Proof We show that for a constant number of unrelated machines, the algorithm given above either concludes that the task system is infeasible, or returns an assignment of

tasks to machines that is feasible with a speedup factor of $1+7\epsilon$. The running time is polynomial in n, given that m and ϵ are constants. First, we redefine $\epsilon = \min\{\epsilon, 1/2\}$.

Suppose that there is a DP-cell of the form $(n, \mathbf{z}, \mathbf{w}, \mathbf{c})$ containing a 'YES'-entry. Due to Lemma 8 there is an assignment $\mathcal{T}$ of all tasks to the machines such that $\left\| \sum_{\tau \in \mathcal{T}_i} v'(i,\tau) \right\|_\infty \leq 1+\epsilon$, for each machine $i \in M$. By Lemma 7 this implies that $\left\| \sum_{\tau \in \mathcal{T}_i} v(i,\tau) \right\|_\infty \leq 1+2\epsilon$. Due to Proposition 6 this implies that $dbf^*_{\mathcal{T},i}((1+\epsilon)^k) \leq (1+2\epsilon)(1+\epsilon)^k$ for each $k \in \mathbb{N}$.

Finally, Lemma 6 implies that the computed task assignment is feasible if the machines run with speed $(1+\epsilon)^2(1+2\epsilon) \leq 1+7\epsilon$ (as $\epsilon < 1/2$).

On the other hand, if all DP-cells of the form $(n, \mathbf{z}, \mathbf{w}, \mathbf{c})$ have a 'NO'-entry, then, by Lemma 9, for any task assignment $\mathcal{T}$ there must be a machine i with $\left\| \sum_{\tau \in \mathcal{T}_i} v'(i,\tau) \right\|_\infty > 1+\epsilon$. By Lemma 7 this yields then that also $\left\| \sum_{\tau \in \mathcal{T}_i} v(i,\tau) \right\|_\infty > 1+\epsilon$. Proposition 6 yields that for this machine i there exists a time instant s which is a power of $(1+\epsilon)$ such that $dbf^*_{\mathcal{T},i}(s) > (1+\epsilon)s$. Finally, (the negation of) Lemma 6 yields that there is a time instant r for which $dbf_{\mathcal{T},i}(r) > r$. Since, for any partition $\mathcal{T}$, there exist a machine i and a time instant r such that $\mathcal{T}$ violates the feasibility condition, it follows that the task system T is infeasible if the machines run at unit speed.

The claim that the running time is polynomial in n for given constant m and ϵ follows from Proposition 7 and the fact that each entry of the table can be decided upon in polynomial time. Herewith, note that $L = O(\log_{(1+\epsilon)}(1/\epsilon^2))$, which is constant if ϵ is constant. □

6 Conclusion

In this paper we presented the first results for assigning sporadic tasks with arbitrary deadlines to unrelated parallel processors. Through the development of a new LP rounding procedure and approximations of the *demand bound function* we found a $8+2\sqrt{6} \approx 12.9$-approximate feasibility test for an arbitrary number of machines. We hope that future research might bring the approximation ratio down and close the gap between 12.9 and our lower bound of 2. One possible tool might be configuration-LPs which are often stronger than assignment-LPs like the one that we use here (see e.g., [4,8,32]). Another interesting direction would be to obtain better approximations for the case that the processors are unrelated but there are only a few types of processors (e.g., CPUs and GPUs), as done in e.g., [29,30,33].

Our rounding procedure is very general and can not only be applied to the problem of assigning tasks to machines but for any assignment problem which allows for a sparse linear program. It would be interesting to see other new applications for it. Additionally, it would be interesting if a better rounding procedure can be given for large values of γ.

For a constant number of machines we give a polynomial-time approximation scheme. While for scheduling jobs on unrelated machines, a PTAS is relatively easy to obtain once one has the tools from Lenstra et al. [26] at hand, for assigning sporadic tasks, much more sophisticated machinery was required.

References

1. Albers, K., Slomka, F.: An event stream driven approximation for the analysis of real-time systems. In: Proceedings of 16th Euromicro Conference on Real-Time Systems, pp. 187–195 (2004)
2. Anand, S., Garg, N., Megow, N.: Meeting deadlines: how much speed suffices? In: Aceto, L., Henzinger, M., Sgall, J. (eds.) Proceedings of 38th International Colloquium on Automata, Languages and Programming, Volume 6755 of Lecture Notes in Computer Science, pp. 232–243 (2011)
3. Andersson, B., Tovar, E.: Competitive analysis of partitioned scheduling on uniform multiprocessors. In: Proceedings of 21st International Parallel and Distributed Processing Symposium, pp. 1–8. IEEE (2007)
4. Asadpour, A., Feige, U., Saberi, A.: Santa Claus meets hypergraph matchings. In: Approximation, Randomization and Combinatorial Optimization: Algorithms and Techniques, Volume 5171 of Lecture Notes in Computer Science, pp. 10–20. Springer, Berlin (2008)
5. Awerbuch, B., Azar, Y., Grove, E.F., Kao, M.-Y., Krishnan, P., Vitter, J.S.: Load balancing in the L_p norm. In: Proceedings of 36th Symposium on Foundations of Computer Science, pp. 383–391. IEEE (1995)
6. Azar, Y., Epstein, A.: Convex programming for scheduling unrelated machines. In: Proceedings of 37th Symposium on Theory of Computing, pp. 331–337. ACM (2005)
7. Baker, T.P., Baruah, S.K.: Schedulability analysis of multiprocessor sporadic task systems. In: Handbook of Real-Time and Embedded Systems, chapter 3. CRC Press (2007)
8. Bansal, N., Sviridenko, M.: The Santa Claus problem. In: Proceedings of 38th Symposium on Theory of Computing, pp. 31–40. ACM, New York, NY, USA (2006)
9. Baruah, S., Fisher, N.: The partitioned multiprocessor scheduling of sporadic task systems. In: Proceedings of 26th IEEE Real-Time Systems Symposium, pp. 321–329. IEEE (2005)
10. Baruah, S., Mok, A., Rosier, L.: Preemptively scheduling hard-real-time sporadic tasks on one processor. In: Proceedings of 11th IEEE Real-Time Systems Symposium, pp. 182–190. IEEE (1990)
11. Baruah, S.K., Bonifaci, V., Marchetti-Spaccamela, A., Stiller, S.: Improved multiprocessor global schedulability analysis. Real-Time Syst. **46**(1), 3–24 (2010)
12. Baruah, S.K., Pruhs, K.: Open problems in real-time scheduling. J. Sched. **13**, 577–582 (2010)
13. Bonifaci, V., Marchetti-Spaccamela, A., Stiller, S.: A constant-approximate feasibility test for multiprocessor real-time scheduling. Algorithmica **62**(3–4), 1034–1049 (2012)
14. Chakraborty, S., Künzli, S., Thiele, L.: Approximate schedulability analysis. In: Proceedings of 23rd IEEE Real-Time Systems Symposium, pp. 159–168. IEEE (2002)
15. Chen, J.-J., Chakraborty, S.: Resource augmentation bounds for approximate demand bound functions. In: Proceedings of 32nd IEEE Real-Time Systems Symposium, pp. 272–281. IEEE (2011)
16. Ebenlendr, T., Krčál, M., Sgall, J.: Graph balancing: a special case of scheduling unrelated parallel machines. In: Proceedings of 19th Symposium on Discrete Algorithms, pp. 483–490 (2008)
17. Eisenbrand, F., Rothvoß, T.: A PTAS for static priority real-time scheduling with resource augmentation. In: Aceto, L., Damgård, I., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) Proceedings of 35th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, vol. 5125, pp. 246–257. Springer, Berlin (2008)
18. Eisenbrand, F., Rothvoß, T.: EDF-schedulability of synchronous periodic task systems is coNP-hard. In: Proceedings of 21st Symposium on Discrete Algorithms, pp. 1029–1034 (2010)
19. Fisher, N., Baruah, S., Baker, T.P.: The partitioned scheduling of sporadic tasks according to static-priorities. In: Proceedings of 18th Euromicro Conference on Real-Time Systems, pp. 118–127 (2006)
20. Jansen, K., Porkolab, L.: Improved approximation schemes for scheduling unrelated parallel machines. In: Proceedings of 31st Symposium on Theory of Computing, pp. 408–417. ACM (1999)
21. Karloff, H.: Linear Programming. Birkhäuser, Basel (1991)
22. Karp, R.M.: Reducibility among combinatorial problems. Complex. Comput. Comput. **40**, 85–103 (1972)
23. Karp, R.M., Leighton, F.T., Rivest, R.L., Thompson, C.D., Vazirani, U.V., Vazirani, V.V.: Global wire routing in two-dimensional arrays. Algorithmica **2**, 113–129 (1987)
24. Anil Kumar, V.S., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: Approximation algorithms for scheduling on multiple machines. In: Proceedings of 46th Symposium on Foundations of Computer Science, pp. 254–263. IEEE (2005)
25. Lau, L.C., Ravi, R., Singh, M.: Iterative Methods in Combinatorial Optimization. Cambridge University Press, Cambridge, MA (2011)

26. Lenstra, J.K., Shmoys, D.B., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. **46**(1–3), 259–271 (1990)
27. Liu, C., Layland, J.: Scheduling algorithms for multiprogramming in a hard real-time environment. J. ACM **20**, 46–61 (1973)
28. Phillips, C.A., Stein, C., Torng, E., Wein, J.: Optimal time-critical scheduling via resource augmentation. Algorithmica **32**, 163–200 (2002)
29. Raravi, G., Andersson, B., Bletsas, K.: Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors. Real-Time Syst. **49**, 29–72 (2013)
30. Raravi, G., Nélis, V.: A PTAS for assigning sporadic tasks on two-type heterogeneous multiprocessors. In: Proceedings of 33rd IEEE Real-Time Systems Symposium, pp. 117–126. IEEE (2012)
31. Shmoys, D.B., Tardos, É.: An approximation algorithm for the generalized assignment problem. Math. Program. **62**(1–3), 461–474 (1993)
32. Svensson, O.: Santa Claus schedules jobs on unrelated machines. J. Comput. **41**(5), 1318–1341 (2012)
33. Wiese, A., Bonifaci, V., Baruah, S.: Partitioned EDF scheduling on a few types of unrelated multiprocessors. Real-Time Syst. **49**(2), 219–238 (2013)
34. Williamson, D.P., Shmoys, D.B.: The Design of Approximation Algorithms. Cambridge University Press, Cambridge, MA (2011)