Server Cloud Scheduling

Marten Maack[1] · **Friedhelm Meyer auf der Heide[1]** · **Simon Pukrop[1]**

Abstract

Consider a set of jobs connected to a directed acyclic task graph with a fixed source and sink. The edges of this graph model precedence constraints and the jobs have to be scheduled with respect to those. We introduce the server cloud scheduling problem, in which the jobs have to be processed either on a single local machine or on one of infinitely many cloud machines. For each job, processing times both on the server and in the cloud are given. Furthermore, for each edge in the task graph, a communication delay is included in the input and has to be taken into account if one of the two jobs is scheduled on the server and the other in the cloud. The server processes jobs sequentially, whereas the cloud can serve as many as needed in parallel, but induces costs. We consider both makespan and cost minimization. The main results are an FPTAS for the makespan objective for graphs with a constant source and sink dividing cut and strong hardness for the case with unit processing times and delays.

1 Introduction

Scheduling with precedence constraints with the goal of makespan minimization is widely considered a fundamental problem. It has already been studied in the 1960s

---

✉ Simon Pukrop
  simonjp@mail.upb.de

  Marten Maack
  martenm@mail.upb.de

  Friedhelm Meyer auf der Heide
  fmadh@mail.upb.de

[1] Department of Computer Science, Heinz Nixdorf Institute, Paderborn University, Paderborn, Germany

by Graham [1] and receives a lot of research attention up to this day (see e.g. [2–4]). One problem variant that has received particular attention recently, is the variant with communication delays (e.g. [4–6]). Another, more contemporary topic concerns scheduling using external resources like, for instance, machines from the cloud and several models in this context have been considered of late (e.g. [7–9]). In this paper, we introduce and study a model closely connected to both settings, where jobs with precedence constraints may either be processed on a single server machine or on one of many cloud machines. Here, communication delays may occur only if the computational setting is changed. The server and cloud machines may behave heterogeneously, i.e., jobs may have different processing times on the server and in the cloud, and scheduling in the cloud incurs costs proportional to the computational load performed in this context. Both makespan and cost minimization is considered. We believe that the present model provides a useful link between scheduling with precedence constraints and communication delays on the one hand and cloud scheduling on the other. There is a shorter published conference version [10] of this paper; Sects. 3, 7 and 8 are new content exclusive to this version.

1.1 Problem

We consider a scheduling problem SCS in which a task graph $G = (\mathcal{J}, E)$ has to be scheduled on a combination of a local machine (server) and a limitless number of remote machines (cloud). The task graph is a directed, acyclic graph with exactly one source $\mathcal{S} \in \mathcal{J}$ and exactly one sink $\mathcal{T} \in \mathcal{J}$. Each job $j \in \mathcal{J}$ has a processing time on the server $p_s(j)$ and on the cloud $p_c(j)$. We consider $p_s(\mathcal{S}) = p_s(\mathcal{T}) = 0$ and $p_c(\mathcal{S}) = p_c(\mathcal{T}) = \infty$. For every other job the values of p_s and p_c can be arbitrary in $\mathbb{N}_0$, meaning that the server and the cloud are unrelated machines in our default model. An edge $e = (i, j)$ denotes precedence, i.e., job i has to be fully processed before job j can start. Furthermore an edge $e = (i, j)$ has a communication delay of $c(i, j) \in \mathbb{N}_0$, which means that after job i finished, j has to wait an additional $c(i, j)$ time steps before it can start, if i and j are not both scheduled on the same type of machine (server or cloud).

A schedule π is given as a tuple $(\mathcal{J}^s, \mathcal{J}^c, C)$. $\mathcal{J}^s$ and $\mathcal{J}^c$ are a proper partition of $\mathcal{J}$: $\mathcal{J}^s \cap \mathcal{J}^c = \emptyset$ and $\mathcal{J}^s \cup \mathcal{J}^c = \mathcal{J}$. The sets $\mathcal{J}^s$ and $\mathcal{J}^c$ denote jobs that are processed on the server or cloud in π, respectively. Lastly, $C : \mathcal{J} \mapsto \mathbb{N}_0$ maps jobs to their completion time.

We introduce some notation before we formally define the validity of a schedule. Let $p^\pi(j)$ be equal to $p_s(j)$ iff $j \in \mathcal{J}^s$, and $p_c(j)$ iff $j \in \mathcal{J}^s$. The value $p^\pi(j)$ denotes the actual processing time of job j in π. Let $E^* := \{(i, j) \in E \mid (i \in \mathcal{J}^s \wedge j \in \mathcal{J}^c) \vee (i \in \mathcal{J}^c \wedge j \in \mathcal{J}^s)\}$ be the set of edges between jobs on different computational contexts (server or cloud). Intuitively, for all the edges in E^* we have to take the communication delays into consideration, for all edges in $E \setminus E^*$ we only care about the precedence.

We call a schedule π valid if and only if the following conditions are met:

(a) There is always at most one job processing on the server:
$\forall_{i \in \mathcal{J}^s} \forall_{j \in \mathcal{J}^s \setminus \{i\}} : (C(i) \leq C(j) - p^\pi(j)) \vee (C(i) - p^\pi(i) \geq C(j))$

(b) Tasks are not started before the preceding tasks have been finished and the required communication is done:

$$\forall_{(i,j)\in E\setminus E^*} : (C(i) \le C(j) - p^{\pi}(j))$$
$$\forall_{(i,j)\in E^*} : (C(i) + c(i,j) \le C(j) - p^{\pi}(j))$$

The makespan ($mspan$) of a schedule is given by the completion time of the sink $C(\mathcal{T})$. The cost ($cost$) of a schedule is given by the time it spends processing tasks on the cloud: $\sum_{i\in\mathcal{J}^c} p^{\pi}(i)$. Note here, that by requiring $p_s(\mathcal{S}) = p_s(\mathcal{T}) = 0$ and $p_c(\mathcal{S}) = p_c(\mathcal{T}) = \infty$, we assume every job to start and end on the server. This is done only for convenience as it defines a clear start and end state for each schedule.

Naturally two different optimization problems arise from the definition. First, given a deadline d, find a schedule with lowest cost and $mspan = C(\mathcal{T}) \le d$. Second, given a cost budget b, find a schedule with smallest makespan and $cost = \sum_{i\in\mathcal{J}^c} p^{\pi}(i) \le b$. In both instances the d, respectively the b, is strict. The natural decision variant is: given both d and b find a schedule that adheres to both, if one exists.

Remark 1 Instances of SCS might contain schedules with a makespan (and therefore cost) of 0. We can check for those in polynomial time: First, remove all edges with communication delay 0, we get a set of connected components K. Iff $\forall_{k\in K} \left(\forall_{j\in k} p_s(j) = 0\right) \vee \left(\forall_{j\in k} p_c(j) = 0\right)$, then there is a schedule with makespan of 0. For the rest of the paper we will assume that our algorithms check that beforehand and are only interested in schedules with $mspan > 0$.

1.2 Results

We start by establishing (weak) NP-hardness already for the case without communication delays and very simple task graphs. More precisely, for the case in which the task graph forms one chain starting with the source and ending with the sink and the case in which the graph is fully parallel, i.e., each job $j \in \mathcal{J} \setminus \{\mathcal{S}, \mathcal{T}\}$ is only preceded by the source and succeeded by the sink. On the other hand, we establish FPTAS results for both the chain and fully parallel case with arbitrary communication delays and with respect to both objective functions. Furthermore, we present a 2-approximation for the case without delays and identical server and cloud machines ($p_c = p_s$) but arbitrary task graph and the makespan objective and show that the respective algorithm can also be used to solve the problem optimally with respect to both objectives in the case of unit processing times. These results are all relatively simple and are discussed in Sect. 2. In Sect. 3 we generalize the previous two task graph models (chain and fully parallel) into one, called extended chain graphs. We present a $(2 + \varepsilon)$-approximation for the budget restrained makespan minimization for this class of task graphs. Furthermore, we discuss some small assumptions on the problem instance, which allow us to achieve FPTAS results instead. We end the section by giving a reduction from the strongly NP-hard $1 \mid r_j \mid \sum w_j U_j$ problem [11]. In Sect. 4 we aim to generalize the previous FPTAS results regarding the makespan as much as possible. We are able to show that an FPTAS can be achieved as long as the *maximum cardinality source and sink dividing cut* ψ is constant. Intuitively, this parameter upper bounds

the number of edges that have to be considered together in a dynamic program and in many relevant problem variants it can be bounded or replaced by the longest anti-chain length. We provide a formal definition in Sect. 4. Next, we turn our attention to strong NP-hardness results in Sect. 5. We are able to show, that a classical reduction due to Lenstra and Rinnooy Kan [12] can be adapted to prove NP-hardness already for the variant of SCS without communication delays and processing times equal to one or two. Now, in the case of unit processing times without communication delays this can be trivially solved in polynomial time, and hence we are interested in the case with unit processing times and communication delays. We design an intricate reduction to show that this very basic case is NP-hard as well. Note that in this setting the server and cloud machines are implicitly identical. Furthermore, we are able to show that a slight variation of this reduction implies that no constant approximation with respect to the cost objective can be achieved regarding the general problem. In Sect. 6, we consider approximation algorithms for the case with unit processing times and delays. We show that a relatively simple approach yields a $\frac{1+\varepsilon}{2\varepsilon}$-approximation for $\varepsilon \in (0, 1]$ regarding the cost objective if we allow a makespan of $(1 + \varepsilon)d$. In Sect. 7, we establish some natural generalizations on the model and sketch how those can be solved by slight adaptations of our algorithms for extended chain and constant ϕ graphs. Lastly, in Sect. 8 we show how to give an α-approximation, for any chosen $\alpha > 0$, on the pareto front of a problem with a task graph with constant ϕ, when we look at the problem as a multi objective optimization problem. This means, that for any point in the actual pareto front, we give a nearby feasible point that is only worse by a factor of $1 + \alpha$ in both dimensions. In Table 1 we give an overview over the important results.

1.3 Related Work

Probably the closest related model to the one considered in this paper was studied by Aba et al. [7]. In this paper the input is very similar, however, in both computational settings an unbounded number of machines may be used and the goal is makespan minimization. The authors show NP-hardness on the one hand, and identify cases that can be solved in polynomial time on the other. In the conclusion of this paper a model very similar to the one studied in this work is mentioned as an interesting research direction. For a detailed discussion of related models, we refer to the preprint version of the above work [7].

The present model is closely related to the classical problem of makespan minimization on parallel machines with precedence constraints, where a set of jobs with processing times, a precedence relation on the jobs (or a task graph), and a set of m machines are given. The goal is to assign the jobs to starting times and machines such that the precedence constraints are met and the last job finishes as soon as possible. In the 1960s, Graham [1] introduced the list scheduling heuristic for this problem and proved it to be a $(2 - \frac{1}{m})$-approximation. Interestingly, to date, this is essentially the best result for the general problem. On the other hand, Lenstra and Rinnooy Kan [12] showed that no better than $\frac{4}{3}$-approximation can be achieved for the problem with unit processing times, unless P=NP. In more recent days, there has been

Table 1 An overview of the results of this paper

Algorithmic results	
Fully parallel or chain task graph	FPTAS w.r.t. cost and makespan
Extended chain task graph	$(2 + \varepsilon)$-approximation w.r.t. makespan
Extended chain + additional assumptions	FPTAS w.r.t. makespan
Extended chain task graph + generalizations	$(4 + \varepsilon)$-approximation w.r.t. makespan
Task graph with constant ψ	FPTAS w.r.t. makespan
Task graph with constant ψ	α-approximation of Pareto front, for any $\alpha > 0$
Task graph with constant ψ + generalizations	FPTAS w.r.t. makespan
$c = 0$, $p_c = p_s$ (no delays, identical machines)	2-approximation w.r.t. makespan
$c = 0$, $p_c = p_s = 1$	Polynomial w.r.t. makespan and cost
$c = p_c = p_s = 1$ (unit delays, unit sizes)	$\frac{1+\varepsilon}{2\varepsilon}$-approximation w.r.t. cost with makespan at most $(1 + \varepsilon)d$

Hardness results	
Fully parallel or chain task graph, $c = 0$	(weakly) NP-hard
Extended chain task graph	(strongly) NP-hard
$\forall j \in \mathcal{J} : c(j) = 0,\ p_c(j), p_s(j) \in \{1, 2\}$	(strongly) NP-hard
$c = p_c = p_s = 1$ (unit delays, unit sizes)	(strongly) NP-hard
General problem	No constant approximation w.r.t. cost

a series of exciting new results for this problem starting with a paper by Svensson [13] who showed that no better than 2-approximation can be hoped for assuming a variant of the unique games conjecture. Furthermore, Levey and Rothvoss [2] presented an approximation scheme with nearly quasi-polynomial running time for the variant with unit processing times and a constant number of machines, and Garg [3] improved the running time to quasi-polynomial shortly thereafter. These results utilized so called LP-hierarchies to strengthen linear programming relaxations of the problems. This basic approach has been further explored in a series of subsequent works (e.g. [4–6]), which in particular also investigate the problem variant where a communication delay is incurred for pairs of precedence-constrained jobs running on different machines. The latter problem variant is closely related to our setting as well.

Lastly, there is at least a conceptual relationship to problems where jobs are to be executed in the cloud. For example, a problem was considered by Saha [8] in which cloud machines have to be rented in fixed time blocks in order to schedule a set of jobs with release dates and deadlines minimizing the costs which are proportional to the rented time blocks. Another example is a work by Mäcker et al. [9] in which machines of different types can be rented from the cloud and machine dependent setup times have to be paid before they can be used. Jobs arrive in an online fashion and the goal is again cost minimization. Both papers reference further work in this context.

2 Preliminary Results: Chains and Fully Parallel

In this section we collect some results that can be considered low hanging fruits and give a first overview concerning the complexity and approximability of our problem. In particular, we show weak NP-hardness already for cases with very simple task graphs and without communication delays. Furthermore, we discuss complementing FPTAS results and a 2-approximation for the case with identical cloud and server machines and without communication delays.

2.1 Hardness

We show that SCS is NP-hard even for two very simple types of taskgraphs and in a case where every communication time is 0. For both of these reductions we use the decision variant of the problem: given both a deadline d and a budget b, find a schedule that satisfies both. Naturally this will show the hardness of both the cost minimization as well as the makespan minimization problem. We start by reducing the decision version of knapsack to SCS with a chain graph as its task graph. The knapsack problem is given as a capacity C, a value threshold V and a set of items $\{1, \ldots, n\}$ with weights w_i and values v_i.

The question is, if there exist is a subset of items S such that $\sum_{i \in S} w_i \leq C$ and $\sum_{i \in S} v_i \geq V$. We create the respective SCS problem as follows. For every item $i \in \{1, \ldots, n\}$ create a task with $p_s(i) = w_i + v_i$ and $p_c(i) = v_i$. Consider a task graph with those tasks as a chain (in an arbitrary order) and each resulting edge (i, j) has $c(i, j) = 0$. We set the deadline to $d = \sum_{1 \leq i \leq n} v_i + C$ and the budget to $b = \sum_{1 \leq i \leq n} v_i - V$. It is left to show, that there is a solution to the knapsack problem if and only if there is a schedule to our transformed problem. Basically we show that there is a one to one relation between our schedules and knapsack solutions. Assume there is some feasible solution (subset of items S) for the knapsack problem with value V'. For each $i \in S$ we put the respective task in $\mathcal{J}^s$ and the rest in $\mathcal{J}^c$. Since the task graph is a chain we can compute a minimal makespan from this partition: $\sum_{1 \leq i \leq n} v_i + \sum_{i \in S} w_i$ which is smaller or equal to d if and only if $\sum_{i \in S} w_i \leq C$. The cost for the schedule is equal to $\sum_{1 \leq i \leq n} v_i - V'$. Therefore, the cost for the schedule is smaller or equal to b exactly when $V' \geq V$. It is easy to see that we can construct a knapsack solution from a schedule in a similar vein, therefore we conclude:

Theorem 1 *The SCS problem is weakly NP-hard for chain graphs and without communication delays.*

Secondly we look at problems with fully parallel task graphs, which means that every job j besides $\mathcal{S}$ and $\mathcal{T}$ has exactly two edges: $(\mathcal{S}, j)$ and $(j, \mathcal{T})$. Here we do a simple partition reduction. Given a set S of natural numbers, the question is, if there is a partition into sets S_1 and S_2 such that $\sum_{i \in S_1} i = \sum_{i \in S_2} i$? For every element i in S we create a task with $p_s(j) = p_c(j) = i$, set $d = b = \frac{1}{2} \sum_{i \in S_1} i$. We arrange the tasks into a fully parallel task graph where each edge (i, j) has $c(i, j) = 0$. Imagine a solution S_1, S_2 for the partition problem. We schedule every task related to an integer in S_1 on the server and every other task on the cloud. Since everything

is fully parallel and there are no communication delays we can conclude a makespan of $\max\{\sum_{i \in S_1} i, \max_{i \in S_2} i\}$ and costs of $\sum_{i \in S_2}$. This is a correct solution for the scheduling problem if and only if $\sum_{i \in S_1} i = \sum_{i \in S_2} i$. Again it is easy to see that an equivalent argument can be made for the other direction.

Theorem 2 *The SCS problem is weakly NP-hard for fully parallel graphs and without communication delays.*

2.2 Algorithms

In the following, we present complementing FPTAS results for the variants of SCS with fully parallel and chain task graphs. Furthermore, in both of the above reductions we did have no communication delays and in one of them the jobs had the same processing time on the server and the cloud. Hence, we take a closer look at this case as well and present a simple 2-approximation even for arbitrary task graphs and with respect to the makespan objective.

2.2.1 Fully Parallel Case

We show that the variant of SCS with fully parallel task graph can be dealt with using straight-forward applications of well-known results and techniques. In particular, we can design two simple dynamic programs for the search version of the problem that consider for each job the two possibilities of scheduling them on the cloud or on the server and compute for each possible budget or deadline the lowest makespan or cost, respectively, that can be achieved with the jobs considered so far. These dynamic programs can then be combined with suitable rounding procedures that reduce the number of considered states and search procedures for approximate values for the optimal cost or makespan, respectively, yielding:

Theorem 3 *There is an FPTAS for SCS with fully parallel task graph with respect to both the cost and the makespan objective.*

Proof We start by designing the dynamic programs for the search version of the problem with budget b and deadline d. Without loss of generality, we assume $\mathcal{J} = \{0, 1, \ldots, n, n+1\}$ with $\mathcal{S} = 0$, $\mathcal{T} = n+1$ and set $c(j) = c(\mathcal{S}, j) + c(j, \mathcal{T})$.

For each deadline $d' \in \{0, 1, \ldots, d\}$ and $j \in \mathcal{J}$, we want to compute the smallest cost $C[j, d']$ of all the schedules of the jobs $0, 1, \ldots, j$ adhering to the deadline d' on the server ($j = 0$ denotes the trivial case that no job after the source has been scheduled). We initialize $C[0, d'] = 0$ for each d'. For all other jobs j we consider the two possibilities of scheduling it on the cloud or server. In particular, let $C_1[j, d'] = C[j-1, d'] + p_c(j)$ if $p_c(j) + c(j) \leq d$ and $C_1[j, d'] = \infty$ otherwise, and, furthermore, $C_2[j, d'] = C[j-1, d' - p_s(j)]$ if $p_s(j) \leq d'$ and $C_2[j, d'] = \infty$ otherwise. Then, we may set $C[j, d'] = \min\{C_1(j, d'), C_2(j, d')\}$. Now, if $C[n+1, d] > b$, we know that there is no feasible solution for the search version, and otherwise we can use backtracking starting from $C[n+1, d]$ to find one. The time and space complexity is polynomial in d and n.

In the second dynamic program, we compute the smallest makespan $M[j, b']$ of all the schedules of the jobs $0, 1, \ldots, j$ adhering to the budget b', for each budget $b' \in \{0, 1, \ldots, b\}$ and $j \in \mathcal{J}$. Again, we set $M[0, b'] = 0$ for each b' and consider the two possibilities of scheduling job j on the cloud or server. To that end, let $M_1[j, b'] = \max\{M[j-1, b' - p_c(j)], p_c(j) + c(j)\}$ if $p_c(j) + c(j) \le d$ and $b' - p_c(j) \ge 0$. Otherwise, set $M_1[j, b'] = \infty$, furthermore, $M_2[j, b'] = M[j-1, b'] + p_s(j)$. Then, we may set $M[j, b'] = \min\{M_1(j, b'), M_2(j, b')\}$. Again, if $M[n+1, b] > d$, we know that there is no feasible solution for the search version, and otherwise we can use backtracking starting from $M[n+1, b]$ to find one. The time and space complexity is polynomial in b and n.

For both programs, we can use rounding and scaling approaches to trade the complexity dependence in d or b with a dependence in $poly(n, \frac{1}{\varepsilon})$ incurring a loss of a factor $(1 + \mathcal{O}(\varepsilon))$ in the makespan or cost, respectively, if a solution is found. This can then be combined with a suitable search procedure for approximate values of the optimal makespan or cost. For details, we refer to Sect. 4, where such techniques are used and described in more detail. In addition to the techniques mentioned there, the possibility of a cost zero solution has to be considered which can easily be done in this case. □

2.2.2 Chain Graph Case

We present FPTAS results for the variant of SCS with chain task graph. The basic approach is very similar to the fully parallel case.

Theorem 4 *There is an FPTAS for SCS with chain task graph with respect to both the cost and the makespan objective.*

Proof We again start by designing dynamic programs for the search version of the problem with budget b and deadline d. Without loss of generality, we assume $\mathcal{J} = \{0, 1, \ldots, n+1\}$ with $\mathcal{S} = 0$, $\mathcal{T} = n+1$, and $j \in \{0, 1, \ldots, n+1\}$ being the j-th job in the chain.

For each deadline $d' \in \{0, 1, \ldots, d\}$, job $j \in \{0, 1, \ldots, n+1\}$, and location $loc \in \{s, c\}$ (referring to the server and cloud) we want to compute the smallest cost $C[d', j, loc]$ of all the schedules of the jobs $1, \ldots, j$ adhering to the deadline d' and with the job j being scheduled on loc. To that end, we set $C[d', 0, s] = 0$, $C[d', 0, c] = \infty$, and with slight abuse of notation use the convention $C[z, j, loc] = \infty$ for $z < 0$. Further values can be computed via the following recurrence relations:

$$C[d', j, s] = \min\{C[d' - p_s(j) - c(j-1, j), c], C[d' - p_s(j), s]\}$$
$$C[d', j, c] = \min\{C[d' - p_c(j), c] + p_c(j), C[d' - p_c(j) - c(j-1, j), s] + p_c(j)\}$$

If $C[d, n+1, s] > b$, we know that there is no feasible solution for the search version, and otherwise we can use backtracking starting from $C[d, n+1, s]$ to find one. The time and space complexity is polynomial in d and n.

In the second dynamic program, we compute the smallest makespan $M[j, b', loc]$ of all the schedules of the jobs $0, \ldots, j$ adhering to the budget b' and with job j placed on location loc, for each $b' \in \{0, 1, \ldots, b\}$, $j \in \{0, 1, \ldots, n+1\}$ and $loc \in \{s, c\}$.

We set $M[b', 0, s] = 0$, $M[b', 0, c] = \infty$, use the convention $M[z, j, loc] = \infty$ for $z < 0$, and the recurrence relations:

$$M[b', j, s] = \min\{M[b', c] + p_s(j) + c(j - 1, j), M[b', s] + p_s(j)\}$$
$$M[b', j, c] = \min\{M[b' - p_c(j), c] + p_c(j), M[b' - p_c(j), s] + p_c(j) + c(j - 1, j)\}$$

If $M[b, n+1, s] > d$, we know that there is no feasible solution for the search version, and otherwise we can use backtracking starting from $M[b, n+1, s]$ to find one. The time and space complexity is polynomial in b and n.

Like in the fully parallel case, we can use rounding and scaling approaches to trade the complexity dependence in d or b with a dependence in $poly(n, \frac{1}{\varepsilon})$ incurring a loss of a factor $(1 + \mathcal{O}(\varepsilon))$ in the makespan or cost, respectively, if a solution is found. This can then be combined with a suitable search procedure for approximate values of the optimal makespan or cost. For details, we refer to Sect. 4, where such techniques are used and described in more detail. In addition to the techniques mentioned there, the possibility of a cost zero solution has to be considered which can easily be done in this case as well. □

3 The Extended Chain Model

As a first step towards more general models we introduce the extended chain model. The main idea here is to find a unifying generalization for the chain and fully parallel case. Informally one can imagine an extended chain as a chain graph where any number of edges were replaced with fully parallel graphs. After giving a formal definition of these graphs we introduce a $(2 + \varepsilon)$-approximation for the budget restrained makespan minimization. That algorithm uses reductions to single machine weighted number of tardy jobs scheduling to solve some intermediate parts via known procedures. Therefore, we briefly discuss this problem here before actually giving our algorithm. We finish the constructive side by exploring some assumptions on problem instances that allow us to achieve FPTAS results with our approach. Lastly, we give a reduction to show that this problem is strongly NP-hard.

3.1 Single Machine Weighted Number of Tardy Jobs

As mentioned before this section reduces some intermediate steps in the algorithm to the single machine weighted tardiness problems, for which we will reuse an already established algorithm.

The single machine weighted number of tardy jobs (WNTJ) problem, or $1 \mid \mid \sum w_j U_j$ in three field notation [14], can be defined as follows: On a single machine, where only one job at a time can be processed, are n jobs to be scheduled. Each job has an integer processing time p_j, weight w_j and due date d_j. A job is called 'late' if it is scheduled completion time $C_j > d_j$ and 'early' if $C_j \leq d_j$. The goal is to find a schedule which minimizes the sum over the weights of the tardy (late) jobs. Pseudo polynomial dynamic programs with runtime in $\mathcal{O}(n \min\{\sum_j p_j, \max_j d_j\})$
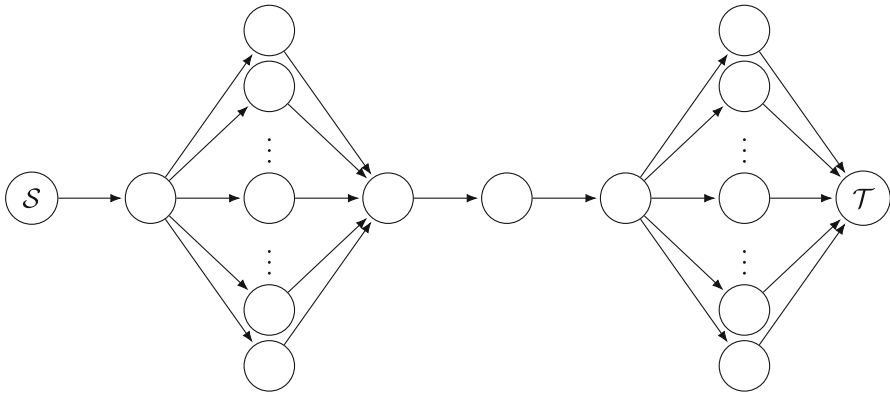
Fig. 1 An example extended chain with two parallel parts

and $\mathcal{O}(n \min\{\sum_j p_j, \sum_j w_j, \max_j d_j\})$, respectively, were given by Lawler and Moore [15] and later Sahni [16]. Denote the former by `wTardyJobs`. For a more comprehensive survey on this (and related) problems, we refer to [17].

3.2 Model

We give a constructive description of extended chain graphs. Let $G = (\mathcal{J}, E)$ with $\mathcal{S} \in \mathcal{J}$ and $\mathcal{T} \in \mathcal{J}$ be a chain graph. For any number of edges $e = (j - 1, j) \in E$ we may remove the edge e and introduce a set of jobs $\mathcal{J}_j$ and for every $j' \in \mathcal{J}_j$ two edges, namely $(j - 1, j')$ and (j', j). The resulting graph $G' = (\mathcal{J}', E')$ is an extended chain graph. We denote by N the total number of jobs (nodes) in the graph. Denote the SCS problem on extended chains by SCS^e. For an example we refer to Fig. 1. Note here, that the introduced subgraphs are fully parallel graphs as described earlier and consequently fully parallel graphs, as well as chain graphs, are a subset of extended chain graphs. This also directly infers that SCS^e is at least weakly NP-hard as shown in Theorems 1 and 2.

3.3 A $(2 + \varepsilon)$-Approximation for Makespan Minimization on the Extended Chain

Theorem 5 *There is a $(2 + \varepsilon)$-approximation algorithm for the budget restrained makespan minimization problem on extended chains.*

We design a pseudo polynomial algorithm, that given a feasible makespan estimate T ($T \geq mspan_{OPT}$) calculates a schedule with makespan at most $\min\{2T, 2mspan_{OPT}\}$. Otherwise ($T < mspan_{OPT}$) the algorithm calculates a schedule with makespan at most $\min\{2T, 2mspan_{OPT}\}$ or no schedule at all. We can use a binary search to find $T \approx OPT$, beginning with the trivial upperbound $T = \sum_{j \in \mathcal{J}'} p_s(j) \geq mspan_{OPT}$

We first introduce notation that follows the constructive description of extended chains above. We assume $\mathcal{J} = \{0, 1, \ldots, n + 1\}$ with $\mathcal{S} = 0$, $\mathcal{T} = n + 1$, and

$j \in \{1, \ldots, n\}$ being the j-th job in the original chain. If there is a parallel subgraph between some jobs $j - 1$ and j we denote the jobs in it by $\mathcal{J}_j = \{0^j, 1^j, \ldots, m^j\}$.

We reuse the state description from Theorem 4, but this time we iteratively create all reachable states by going over the jobs $\{0, 1, \ldots, n + 1\}$. A state is a combination of timestamp $t \in \{0, 1, \ldots, T\}$, job $j \in \{0, 1, \ldots, n + 1\}$, and location $loc \in \{s, c\}$ (referring to server and cloud respectively). The value of a state is the smallest cost of all the schedules of the jobs $0, 1, \ldots, j$ finishing processing during or before timestamp t, with j being scheduled on loc, denoted by $[t, j, loc] = cost$. Note, that we have not mentioned the parallel subgraphs in the description above. We start with the trivial start state $[0, 0(= \mathcal{S}), s] = 0$

Let STATELIST^{j-1} be the list of states for some job of the chain $j - 1$. We create STATELIST^j in the following way: First we create a set of state extensions EXTENSIONS^j, each of form $[\Delta t, loc_{j-1} \to loc_j] = cost$. Then we form every (fitting) combination of a state from STATELIST^{j-1} with an extension from EXTENSIONS^j, which forms STATELIST^j. Lastly we cull all dominated states from STATELIST^j and continue with $j + 1$.

Calculate EXTENSIONS^j:

1. If there is no parallel subgraph between $j - 1$ and j we can simply enumerate all state extensions:

   (a) $j - 1$ on server, j on server: $[p_s(j), \ s \to s] = 0$
   (b) $j - 1$ on server, j on cloud: $[p_c(j) + c(j - 1, j), \ s \to c] = p_c(j)$
   (c) $j - 1$ on cloud, j on server: $[p_s(j) + c(j - 1, j), \ c \to s] = 0$
   (d) $j - 1$ on cloud, j on cloud: $[p_c(j), \ c \to c] = p_c(j)$

2. Otherwise, there is a parallel subgraph between $j - 1$ and j with jobs $\mathcal{J}_j = \{0^j, 1^j, \ldots, m^j\}$.

   (a) $j - 1$ on server, j on server:
       Set $\Delta^{max} = \min\{\sum_{j' \in \mathcal{J}_j} p_s(j'), T\}$, for every Δ^i in $\{0, \ldots, \Delta^{max}\}$, do the following: Set $\mathcal{J}^s = \emptyset$ and $\mathcal{J}^c = \emptyset$. For every $j' \in \mathcal{J}_j$ check:
       - $p_s(j') > \Delta^i$ and $c(j - 1, j') + p_c(j') + c(j', j) > \Delta^i$:
         break and go to next Δ^i (state extension $[\Delta^i, \ s \to s]$ not feasible)
       - $p_s(j') > \Delta^i$ and $c(j - 1, j') + p_c(j') + c(j', j) \le \Delta^i$:
         add j' to $\mathcal{J}^c$ (j' has to be put on the cloud)
       - $p_s(j') \le \Delta^i$ and $c(j - 1, j') + p_c(j') + c(j', j) > \Delta^i$:
         add j' to $\mathcal{J}^s$ (j' has to be put on the server)

       If $\sum_{j' \in \mathcal{J}^s} p_s(j') > \Delta^i$ break and go to next Δ^i. Create a WNTJ instance as follows: For every job $j' \in \mathcal{J}_j \setminus (\mathcal{J}^s \cup \mathcal{J}^c)$ create a job j'' with processing time $p_{j'} = p_s(j')$, deadline $d_{j''} = \Delta^i - \sum_{j' \in \mathcal{J}^s} p_s(j')$ and weight $w_{j''} = p_c(j')$. Solve this problem with `wTardyJobs`, let V be the cost of the solution. Add $[\Delta^i, \ s \to s] = \sum_{j' \in \mathcal{J}^c} p_c(j') + V$ to EXTENSIONS^j. (Remark: This could also be solved as a knapsack problem, but we need WNTJ later either way.)
   (b) $j - 1$ on server, j on cloud:

Set $\Delta^{max} = \min\{\sum_{j' \in \mathcal{J}_j} p_s(j') + \max_{j' \in \mathcal{J}_j} c(j', j), T\}$, for every Δ^i in $\{0, \ldots, \Delta^{max}\}$, do the following: Set $\mathcal{J}^s = \emptyset$ and $\mathcal{J}^c = \emptyset$. For every $j' \in \mathcal{J}_j$ check:

- $p_s(j') + c(j', j) > \Delta^i$ and $c(j - 1, j') + p_c(j') > \Delta^i$:
  break and go to next Δ^i (state extension $[\Delta^i, s \rightarrow c]$ not feasible)
- $p_s(j') + c(j', j) > \Delta^i$ and $c(j - 1, j') + p_c(j') \leq \Delta^i$:
  add j' to $\mathcal{J}^c$ (j' has to be put on the cloud)
- $p_s(j') + c(j', j) \leq \Delta^i$ and $c(j - 1, j') + p_c(j') > \Delta^i$:
  add j' to $\mathcal{J}^s$ (j' has to be put on the server)

Create a WNTJ instance as follows: For every job $j' \in \mathcal{J}_j \backslash \mathcal{J}^c$ create a job j'' with processing time $p(j'') = p_s(j')$, deadline $d_{j''} = \Delta^i - c(j', j)$ and weight $w_{j''} = p_c(j')$ if $j' \notin \mathcal{J}^s$, $w_{j''} = \infty$ otherwise. Solve this problem with wTardyJobs, let V be the cost of the solution, if $V = \infty$ break. Otherwise, add $[\Delta^i, s \rightarrow c] = \sum_{j' \in \mathcal{J}^c} p_c(j') + V$ to EXTENSIONS^j.

(c) $j - 1$ on cloud, j on server:
This works analogously to the previous case. Simply replace each instance of $c(j', j)$ by $c(j - 1, j')$ and vice versa. Add the resulting extensions to EXTENSIONS^j. Note, that for the reduction there is no computational difference between common release date and different deadlines and different release dates but common deadline.

(d) $j - 1$ on cloud, j on cloud:
We 2-approximates the resulting extensions, by precisely handling the communication to the server, but upperbounding the communication from the server. Repeat case 2b with the two following changes:
For the checks before the problem conversion use $c(j-1, j') + p_s(j') + c(j', j)$ and $p_c(j')$ instead of $p_s(j') + c(j', j)$ and $c(j - 1, j') + p_c(j')$, respectively. Let $\mathcal{J}^{s'} \subseteq \mathcal{J}_j$ be the set of jobs actually put on the server in this step. Add $[\Delta^i + \max_{j' \in \mathcal{J}^{s'}} c(j - 1, j'), c \rightarrow c] = \sum_{j' \in \mathcal{J}^c} p_c(j') + V$ instead of $[\Delta^i, c \rightarrow c] = \sum_{j' \in \mathcal{J}^c} p_c(j') + V$ to EXTENSIONS^j. We wait for the biggest communication delay to pass until we schedule the first job on the server. Note, that $\Delta^i + \max_{j' \in \mathcal{J}^{s'}} c(j', j) \leq 2\Delta^i$ by construction.

For every pair of a state $([t, j - 1, loc] = cost) \in$ STATELIST^{j-1} and $([\Delta t, loc_{j-1} \rightarrow loc_j] = cost') \in$ EXTENSIONS^j with $loc = loc_{j-1}$ add $[t + \Delta t, j, loc_j] = cost + cost'$ to STATELIST^j. After that process, for every triple t, j, loc that has multiple states in STATELIST^j keep only the state with the lowest cost. We can also discard states with $cost > b$ and timestamp $t > 2T$. Repeat this process with $j \rightarrow j + 1$ until we computed STATELIST^{n+1}, simply move through that list and select the state with lowest timestamp t. If there is no such state, there exist no schedule with makespan smaller or equal to T.

Lemma 1 *Given a feasible T, the described procedure calculates a 2-approximation on the optimal makespan in time $poly(N, T)$*

Proof We start by showing the approximation factor. Assume that we added $[\Delta^i, \ c \rightarrow c] = \sum_{j' \in \mathcal{J}^c} p_c(j') + V$ instead of $[\Delta^i + \max_{j' \in \mathcal{J}^{s'}} c(j', j), \ c \rightarrow c] = \sum_{j' \in \mathcal{J}^c} p_c(j') + V$ in step 2d above. That hypothetical algorithm would calculate a (possibly infeasible) solution with makespan $mspan_{ALG}^{hypo} \leq mspan_{OPT}$, since step 2d underestimates the needed time, and everything else is calculated precisely. The actual algorithm has makespan $mspan_{ALG} \leq 2mspan_{ALG}^{hypo}$ and therefore also $mspan_{ALG} \leq 2mspan_{OPT}$.

We show the runtime of the algorithm by bounding the time needed for each iteration of: (1) constructing state extensions EXTENSIONS^j, (2) combining the extensions with the previous STATELIST^{j-1} and (3) culling duplicates from the resulting STATELIST^j.

1. For directly connected jobs $j - 1$ and j we can trivially calculate the 4 options in constant time. Therefore, we are interested in the runtime of steps 2a, 2b, 2c and 2d for some parallel subgraph with jobs $\mathcal{J}_j$. The steps get repeated for Δ^i in $\{0, \ldots, \Delta^{max}\}$, where $\Delta^{max} < T$. The preprocessing in each iteration of all four steps, needs time linear in the size of $\mathcal{J}_j$. Using wTardyJobs in the steps needs time in $\mathcal{O}(|\mathcal{J}_j| \min\{\sum_{j' \in \mathcal{J}_j \setminus \mathcal{J}^c} p_s(j'), \max_{j' \in \mathcal{J}_j \setminus \mathcal{J}^c} d_{j''}\}) \leq \mathcal{O}(T \cdot N^2)$. Overall we need time in $poly(T, N)$ to calculate EXTENSIONS^j, with $|\text{EXTENSIONS}^j| \leq \mathcal{O}(T)$

2. STATELIST^{j-1} contains at most $2T \cdot (n + 2) \cdot 2$ (timestamp, job, location) different states (after the previous culling). We may simply bruteforce all possible combinations from STATELIST$^{j-1} \times$ EXTENSIONS^j. Since both of these sets have at most $poly(T, N)$ elements, the resulting set STATELIST^j also has polynomial size.

3. By culling states from STATELIST^j we reduce it back to size at most $2T \cdot (n + 2) \cdot 2$. It should be obvious, that we can identify duplicate states in polynomial time.

Note that we iterate the above steps for each job $j \in \{1, \ldots, n + 1\}$. Therefore we have a polynomial repetition of steps needing polynomial time. Note that we prevent exponential build-up in the state lists, by culling duplicates after each iteration. □

Now we have to scale our instance, such that our pseudo polynomial algorithm runs in proper polynomial time. For that, we scale T and all p_c, p_s and c by $\frac{N\varepsilon'}{T}$ and round down to the next integer. Then, we run our algorithm with the scaled values, but still use the unscaled p_c to calculate the *value (cost)* of states, as those calculations only factor logarithmically in the runtime, a p_c exponential in the input size is fine. The algorithm now needs time in $poly(N, \lfloor \frac{T \cdot N\varepsilon'}{T} \rfloor) \leq poly(N, \varepsilon')$ and finds a 2 approximation for the scaled instance (given a feasible T). After scaling back up each job and communication delay might need up to $\frac{T}{N\varepsilon'}$ additional time, delaying our whole schedule by at most $3N \cdot \frac{T}{N\varepsilon'} \leq 3\varepsilon'T$. For $\varepsilon = 3\varepsilon'$ and $T = mspan_{OPT}$ our resulting schedule has a makespan of $mspan_{ALG} \leq 2mspan_{OPT} + \varepsilon T = (2 + \varepsilon)mspan_{OPT}$. Via a binary search we can find such a T by repeating our procedure at most $\log \sum_{j \in \mathcal{J}'} p_s(j)$ times. This concludes the proof of Theorem 5.

Corollary 1 *There is a polynomial algorithm for the deadline restrained cost mini-mization problem on extended chains, that finds a schedule with at most optimal cost, but a makespan of $(2 + \varepsilon)d$.*

3.4 Cases with FPTAS

We reconsider the approximation result for three assumptions on the model which allow us to improve the result. Looking back at Theorem 5, we build an algorithm that would be an FPTAS if it were not for case 2d where we needed to double our time frame Δ^i to fit the unaccounted communication delay. In the following part we will only describe how to approach that case, since everything else can stay as it was.

First we assume locally small delays in the parallel subgraphs, meaning that the smallest processing time in the subgraph is at least as big as the largest communication delay. More precisely, for every $\mathcal{J}_e$ with $e = (j - 1, j)$ it holds that

$$\min_{j' \in \mathcal{J}_e} \min\{p_s(j'), p_c(j')\} \geq \max_{j' \in \mathcal{J}_e} \max\{c((j - 1, j')), c((j', j))\}.$$

In this case only the first j^α, and the last job j^ω to be processed on the server are actually affected by their communication delay, since all other delays fit in the time frame, where j^α and j^ω are processed. After the preprocessing of a given Δ^i, for each pair of jobs $j^\alpha, j^\omega \in \mathcal{J}_j \setminus \mathcal{J}^c$ with $j^\alpha \neq j^\omega$ fo the following: Assume j^α, j^ω are the first and last job to be processed on the server, respectively. Add j^α and j^ω to $\mathcal{J}^s$. Now create the WNTJ instance as follows: For every job $j' \in \mathcal{J}_j \setminus (\mathcal{J}^s \cup \mathcal{J}^c)$ create a job j'' with processing time $p_{j'} = p_s(j')$, deadline $d_{j''} = \Delta^i - (c(j - 1, j^\alpha) + c(j^\omega, j)) - \sum_{j' \in \mathcal{J}^s} p_s(j')$ and weight $w_{j''} = p_c(j')$. Solve this problem with wTardyJobs, let V be the cost of the solution and note $[\Delta^i, c \to c]_{j^\omega}^{j^\alpha} = \sum_{j' \in \mathcal{J}^c} p_c(j') + V$. After all $(\mathcal{O}(N^2))$ combinations have been tested, add the smallest $[\Delta^i, c \to c]_{j^\omega}^{j^\alpha}$ to Extensions^j.

Secondly, we assume a constant upper bound c_{max} on the communication delays inside parallel subgraphs. More precisely, for every $\mathcal{J}_e$ with $e = (j - 1, j)$ it holds that

$$c_{max} \geq c(j - 1, j') \text{ and } c_{max} \geq c(j', j).$$

Instead of brute forcing only a first and last job, we brute force the first and last c_{max} time steps. Trivially, jobs with $p_s = 0$ can be put on the server, and therefore there are at most $\mathcal{O}(N_{max}^c \cdot N_{max}^c)$ combinations we have to work through. The remaining part works analogously to the first case.

Lastly, we assume that each job produces some output, that has to be send to all of its direct successors in full, meaning that all outgoing communication delays of a job are equivalent. More precisely, for every $\mathcal{J}_e$ with $e = (j - 1, j)$ it holds that

$$\forall j', j'' \in \mathcal{J}_e : c(j - 1, j') = c(j - 1, j'').$$

Here we can simply reuse the result from step 2b, but subtract $c(j - 1, j')$ from the Δ^i used in the WNTJ problem. Since all $c(j - 1, j')$ are equal, no job could be processed on the server in the first $c(j - 1, j')$ time steps, and all jobs are available after those $c(j - 1, j')$ time steps.

All these, in combination with the previously described scaling approach, lead to FPTAS results:

Theorem 6 *There is an FPTAS for the budget restrained makespan minimization problem on extended chains, if at least one of the following holds for every parallel subgraph $\mathcal{J}_e$ with $e = (j - 1, j)$:*

1. $\min_{j' \in \mathcal{J}_e} \min\{p_s(j'), p_c(j')\} \geq \max_{j' \in \mathcal{J}_e} \max\{c((j - 1, j')), c((j', j))\}$
2. $c_{max} \geq c(j - 1, j')$ *and* $c_{max} \geq c(j', j)$
3. $\forall j', j'' \in \mathcal{J}_e : c(j - 1, j') = c(j - 1, j'')$

3.5 Strong NP-Hardness of Scheduling Extended Chains

As already noted, this problem is at least weakly NP-hard, following from Theorem 1 as well as Theorem 2. We show that this problem is actually strongly NP-hard, by giving a reduction from the strongly NP-hard $1 \mid r_j \mid \sum w_j U_j$ problem [11]. As in Sect. 2.1 we use decision variants of the considered problems, resulting in results for both deadline restrained cost reduction and budget restrained makespan minimization.

Theorem 7 *The SCS^e problem is strongly NP-hard.*

Proof $1 \mid r_j \mid \sum w_j U_j$ is defined as follows: Given a set of jobs $\mathcal{J} = \{1, \ldots, n\}$, each with processing time p_j, release date r_j, deadline d_j and weight w_j, schedule the jobs (without preemption) on a single machine, such that the sum of weights of late jobs is smaller or equal to a given b ($\sum w_j U_j \leq b$). A job j is late ($U_j = 1$) if it finishes processing after d_j, $U_j = 0$ otherwise.

Given an instance of $1 \mid r_j \mid \sum w_j U_j$, create the following decision version of SCS^e. Note that we will substitute "an edge (j, j') with communication delay $c(j, j') = k$" simply by "an edge $c(j, j') = k$" to keep this readable. As per definition create $\mathcal{S}$ and $\mathcal{T}$ with $p_s(\mathcal{S}) = p_s(\mathcal{T}) = 0$ and $p_c(\mathcal{S}) = p_c(\mathcal{T}) = \infty$. Create jobs j^{pre} and j^{post} with $p_s(j^{pre}) = p_s(j^{post}) = \infty$ and $p_c(j^{pre}) = p_c(j^{post}) = 0$ and edges $c(\mathcal{S}, j^{pre}) = 0$ and $c(j^{post}, \mathcal{T}) = 0$. Set $w^{max} = \max_{j \in \mathcal{J}} w_j$ and $d^{max} = \max_{j \in \mathcal{J}} d_j$. For every $j \in \mathcal{J}$ create a job j' with $p_s(j') = p_j$, $p_c(j') = w_j$ and edges $c(j^{pre}, j') = r_j$, $c(j', j^{post}) = w^{max} + d^{max} - d_j$. Set the deadline to $d' = w^{max} + d^{max}$ and the budget $b' = b$. Trivially, in all schedules $\mathcal{S}$ and $\mathcal{T}$ are scheduled on the server, j^{pre} and j^{post} on the cloud. Note that neither of these jobs contributes processing time to the resulting schedule. For better comprehension we give an example of the structure in Fig. 2.

It remains to show, that there is a schedule with $\sum w_j U_j \leq b$ for the original $1 \mid r_j \mid \sum w_j U_j$ problem, iff there is a schedule with cost $\leq b'$ and makespan $\leq d'$ for the SCS^e problem.

Assume that there is a schedule with $\sum w_j U_j \leq b$. We can partition the jobs into two sets $\mathcal{J}^{early}$ and $\mathcal{J}^{late}$, which contain all jobs that are on time or late, respectively.
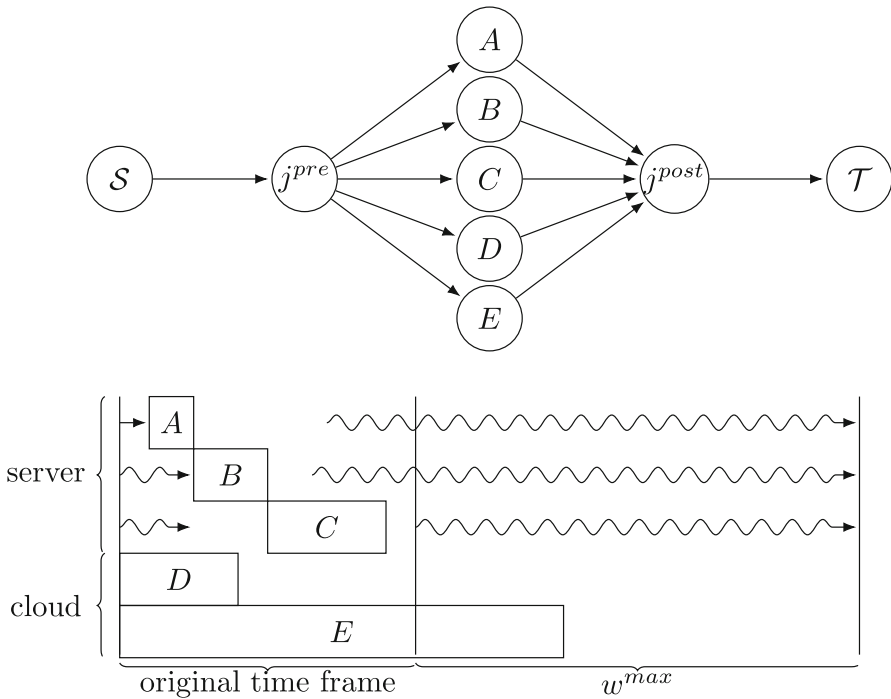
Fig. 2 Schematic example of resulting SCS^e problem for 5 jobs, squiggly arrows represent communication delays and model release dates and deadlines

Place all jobs that correspond to a job from $\mathcal{J}^{late}$ on the cloud and start them immediately. All of them finish before $d' = w^{max} + d^{max}$, since $w^{max} \geq p_c(j')$. Place all remaining jobs ($\mathcal{J}^{early}$) on the server and let them start at the same time as in the original schedule. Since no job starts before its release date no communication delay is violated in the new schedule. Since all jobs from $\mathcal{J}^{early}$ end before their deadline, no communication delay hinders us from scheduling j^{post} and $\mathcal{T}$ at $d' = \Delta^{max} + d^{max}$. The cost of that schedule is equal to the value of $\sum w_j U_j$ in the original schedule and therefore $\leq b$. One can confirm that the other direction works analogously by keeping the schedule of jobs on the cloud intact, and simply processing all jobs from the cloud after that schedule in any order. $\qquad\square$

With argumentation similar to the reduction above, one can show that the $1 \mid r_j \mid \sum w_j U_j$ problem is embedded in step 2d of this chapter's algorithm. This leads to the observation, that we might be able to use approximation results for $1 \mid r_j \mid \sum w_j U_j$ to improve our handling of that case. Sadly, to the best of our knowledge, no approximation algorithms with a provable approximation factor are known for this problem. There are however practical algorithms, which have been tested empirically. Used approaches contain mixed integer programming [18], genetic algorithms [19] and branch-and-bound algorithms [20]. For more information we again refer to [17].
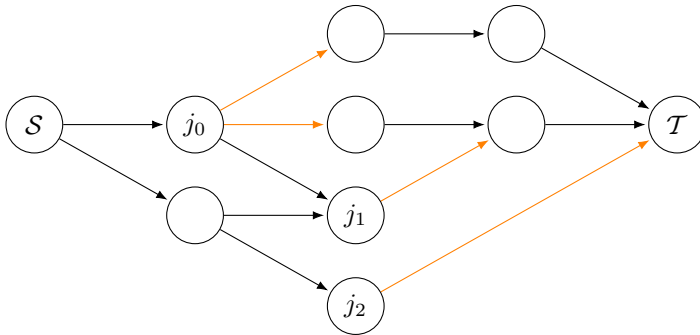
Fig. 3 Example state of a running schedule, *open edges* are orange, loc_{j_i} and f_{j_i} kept for j_0, j_1 and j_2

4 Constant Cardinality Source and Sink Dividing Cut

We introduce the concept of a *maximum cardinality source and sink dividing cut*. For $G = (\mathcal{J}, E)$, let $\mathcal{J}_S$ be a subset of jobs, such that $\mathcal{J}_S$ includes S and there are no edges (j, k) with $j \in \mathcal{J} \setminus \mathcal{J}_S$ and $k \in \mathcal{J}_S$. In other words, in a running schedule $\mathcal{J}_S$ and $\mathcal{J} \setminus \mathcal{J}_S$, could represent already processed jobs and still to be processed jobs respectively. Denote by $\mathcal{J}_S^G$ the set of all such sets $\mathcal{J}_S$. We define

$$\psi := \max_{\mathcal{J}_S \in \mathcal{J}_S^G} |\{(j, k) \in E \mid j \in \mathcal{J}_S \wedge k \in \mathcal{J} \setminus \mathcal{J}_S\}|,$$

the maximum number of edges between any set $\mathcal{J}_S$ and $\mathcal{J} \setminus \mathcal{J}_S$ in G. In a series–parallel task graph ψ is equal to the maximum anti-chain size of the graph.

In this chapter we discuss how to solve or approximate SCS problems with a constant size ψ, but otherwise arbitrary task graphs. We first consider the deadline confined cost minimization, in Theorem 9 we show how to adapt this to the budget confined makespan minimization. We give a dynamic program to optimally solve instances of SCS with arbitrary task graphs. At first we will not confine the algorithm to polynomial time. Consider a given problem instance with $G = (\mathcal{J}, E)$, its source S and sink T, processing times $p_s(j)$ and $p_c(j)$ for each $j \in \mathcal{J}$, communication delays $c(i, j)$ for each $(i, j) \in E$ and a deadline d.

We define intermediate states of a (running) schedule, as the states of our dynamic program (see Fig. 3). Such a state contains two types of variables. First we have two global variables, the timestamp t and the number of time steps the server has been unused f_s. In other words, the server has not finished processing a job since $t - f_s$. The second type is defined per *open edge*. An open edge is a $e = (j, k)$ where j has already been processed, but k has not. For each such edge add the variables $e = (j, k)$ (the edge itself), $loc_j \in \{s, c\}$ denoting if j was processed on the server (s) or the cloud (c) and f_j denoting the number of time steps that have passed since j finished processing. If a job j is contained in multiple open edges, loc_j and f_j are still only included once. Write the state as $[t, f_s, e^1 = (j^1, k^1), loc_{j^1}, f_{j^1}, \ldots, e^m = (j^m, k^m), loc_{j^m}, f_{j^m}]$, where $e^1, \ldots, e^m$ denote all open edges. Note here, that there is

information that we purposefully drop from a state: the completion time and location of every processed job without open edges, as those are not important for future decisions anymore. There might be multiple ways to reach a specific state, but we only care about the minimum possible cost to achieve that state, which is the *value* of the state.

We iteratively calculate the value of every reachable state with $t = 0, 1, 2, \ldots$. We start with the trivial state $[t = 0, f_s = 0, e^1, \ldots, e^m, loc_{\mathcal{S}} = s, f_{\mathcal{S}} = 0] = 0$, where $e^1, \ldots, e^m \in E$ with $e^i = (\mathcal{S}, j)$. This state forms the beginning of our *(sorted) state list*. We keep this list sorted in an ascending order of state values (costs) at all times. We exhaustively calculate every state that is reachable during a specific time step, given the set of states reachable during the previous time step. Intuitively, we try every possible way to "fill up" the still undefined time windows f_s and f_j.

Finally, we give the actual dynamic program in Algorithm 1. After the dynamic program finished, we iterate through the state list one last time and take the first state $[t = d, f_s]$. The value of that state is the minimum cost possible to schedule G in time d. One can easily adapt this procedure to also yield such a schedule, by keeping a list of all processed jobs per state containing their location and completion time.

Lemma 2 DP$_F$GG's *runtime is bounded in* $\mathcal{O}(d^{2\psi+3} \cdot n^{2\psi+1})$.

Proof At any point there are a maximum of $\mathcal{O}(d \cdot (d \cdot n)^{\psi})$ states in the state list. For every t we look at every state. Since we never insert a state in front of the state we are currently inspecting (costs can only increase), this traverses the list exactly once. For each of those states we calculate every possible successor, of which there are $\mathcal{O}(\psi)$ and traverse the state list an additional time to correctly insert or update the state. We iterate from $t = 0$ to d and therefore get a runtime of: $\mathcal{O}(d \cdot ((d \cdot (d \cdot n)^{\psi}) \cdot \psi \cdot (d \cdot (d \cdot n)^{\psi}))) = \mathcal{O}(d^3 \cdot n \cdot (d \cdot n)^{2\psi}) \leq \mathcal{O}(d^{2\psi+3} \cdot n^{2\psi+1})$. $\square$

Algorithm 1 DPFGG: Dynamic Program for General Graphs

1: initialize state list SL with start state (as defined above)
2: **for all** $state \in SL$ **do**
3:     let $\mathcal{J}^{state}$ be the set of all jobs that are endpoints in open edges from $state$
4:     **for all** $j \in \mathcal{J}^{state}$ **do**
5:       **if** $\forall (k, j) \in E : (k, j)$ also open edge in $state$ **then**
6:         *can j be processed on the server?*
7:         **if** $f_s \geq p_s(j)$ **then**
8:           $jFits \leftarrow TRUE$
9:           **for all** $(k, j) \in E$ **do**
10:             **if** $loc_k = s \wedge f_k < p_s(j)$ or $loc_k = c \wedge f_k < p_s(j) + c(k, j)$ **then**
11:               $jFits \leftarrow FALSE$
12:             **end if**
13:           **end for**
14:           **if** $jFits = TRUE$ **then**
15:             *calculate resulting state $state'$, value equal to $state$*
16:             **for all** $(k, j) \in E$ **do**
17:               remove (k, j) from $state'$
18:               **if** j is last open successor of k **then**
19:                 remove f_k and loc_k from $state'$
20:               **end if**
21:               add $f_j = 0, loc_j = s$ and all new open edges to $state'$
22:             **end for**
23:             **if** $state' \in SL$ **then**
24:               update value of $state'$ in SL if new value lower
25:               then move $state'$ to correct position in SL
26:             **else**
27:               add $state'$ to correct position in SL (always after $state$)
28:             **end if**
29:           **end if**
30:         **end if**
31:         *can j be processed on the cloud?*
32:         *analogously to the previous case, cost value of $state'$ increased by $p_c(j)$*
33:       **end if**
34:     **end for**
35: **end for**
36: *check end condition*
37: **if** a state $[t = d, f_s] \in SL$ **then**
38:     return lowest value of such states
39: **end if**
40: **if** $t < d$ **then**
41:     *move from t to $t + 1$*
42:     **for all** each $state \in SL$ **do**
43:       increase t, f_s and each f_j in $state$ by 1
44:     **end for**
45:     Back to step 2
46: **end if**

4.1 Rounding the Dynamic Program

We use a rounding approach on DPFGG to get a program that is polynomial in $n = |\mathcal{J}|$, given that ψ is constant. We scale d, c, p_c, and p_s by a factor $\varsigma := \frac{\varepsilon \cdot d}{2n}$. Denote by $\hat{d} := \lceil \frac{d}{\varsigma} \rceil \leq \frac{2n}{\varepsilon} + 1$, $\hat{p}_s(j) := \lfloor \frac{p_s(j)}{\varsigma} \rfloor$, $\hat{p}_c(j) := \lfloor \frac{p_c(j)}{\varsigma} \rfloor$ and $\hat{c}(x) := \lfloor \frac{c(x)}{\varsigma} \rfloor$. Note

here, that we round up d but everything else down. We run the dynamic program with the rounded values, but still calculate the cost of a state with the original unscaled values.

We transform the output π' to the unscaled instance, by trying to start every job j at the same (scaled back up) point in time as in the scaled schedule. Since we rounded down, there might now be points in the schedule where a job j can not start at the time it is supposed to. This might be due to the server not being free, a parent node of j that has not been fully processed or an unfinished communication delay. We look at the first time this happens and call the mandatory delay on j Δ and increase the start time of every remaining job by Δ. Repeat this process until all jobs are scheduled. We introduce no new conflicts with this procedure, since we always move everything together as a block. Call this new schedule π.

Theorem 8 *Assuming a constant number ψ DPFGG combined with the scaling technique finds a schedule π with at most optimal cost and a makespan $\leq (1+\varepsilon)\cdot d$ in time $poly(n, \frac{1}{\varepsilon})$, for any $\varepsilon > 0$.*

Proof We start by proving the runtime of our algorithm. We can scale the instance in polynomial time, this holds for both scaling down and scaling back up. The dynamic program now takes time in $\mathcal{O}(\hat{d}^{2\psi+3}\cdot n^{2\psi+1})$, where $\hat{d} \leq \frac{2n}{\varepsilon}+1$. Since ψ is constant this results in an dynamic program runtime in $poly(n, \frac{1}{\varepsilon})$. In the end we transform the schedule as described above, for that we go trough the schedule once and delay every job no more than n times. Trivially, this can be done in polynomial time as well.

Secondly we show that the makespan of π is at most $(1+\varepsilon)\cdot d$. Every valid schedule for the unscaled problem is also valid in the scaled problem, meaning that there is no possible schedule we overlook due to the scaling. In the other direction this might not hold. First, while scaling everything down we rounded the deadline up. This means, that scaled back we might actually work with a deadline of up to $d+\varsigma$. Secondly, we had to delay the start of jobs to make sure that we only start jobs when it is actually possible. In the worst case we delay the sink $\mathcal{T}$ a total of $n-2$ times, once for every job other than $\mathcal{S}$ and $\mathcal{T}$. Each time we delay all remaining jobs we can bound the respective $\Delta < 2\cdot\varsigma$. This is due to the fact that each of the delaying options can not delay by more than ς (as that is the maximum timespan not regarded in the scaled problem) and only a direct predecessor job and the communication from it needing longer can coincide to a non-parallel delay. Taking both of these into account, a valid schedule for the scaled problem might use time up to

$$d + \varsigma + (n-2)\cdot(2\varsigma) \leq d + 2n\varsigma = (1+\varepsilon)\cdot d$$

in the unscaled instance.

Lastly, we take a look at the cost of π. While rounding, we did not change the calculation of a states value, and with every valid schedule of the unscaled instance being still valid in the scaled instance we can conclude that the cost of π is smaller or equal to an optimal solution of the original problem. $\square$

Theorem 9 *DPFGG combined with the scaling technique and a binary search over the deadline yields an FPTAS for the cost budget makespan problem, for graphs with a constant number ψ.*

Proof Theorem 8 can be adapted to solve this, assuming that we know a reasonable makespan estimate of an optimal solution to use in our scaling factor. During the algorithm discard any state with costs bigger than the budget and terminate when the first state $[t, f_s]$ is reached. The t gives us the makespan.

Using a makespan estimate that is too big will lead to a rounding error that is not bounded by $\varepsilon \cdot mspan_{OPT}$, a too small estimate might not find a solution. To solve this, we start with an estimate that is purposefully large. Let $d^{max} = \sum_{j \in \mathcal{J}} p_s(j)$ be the sum over all processing times on the server. There is always a schedule with 0 costs and makespan d^{max}. We run our algorithm with the scaling factor $\varsigma^0 := \frac{\varepsilon \cdot d^{max}}{4n}$. Iteratively repeat this process with scaling factor $\varsigma^i = \frac{1}{2^i}\varsigma^0$ for increasing i starting with 1. At the same time half the original deadline estimate in each step, which leads to $\hat{d}$, and therefore the runtime, to stay the same in each iteration. End the process when the algorithm does not find a solution for the current i and deadline estimation. This infers that there is no schedule with the wanted cost budget and a makespan smaller or equal to $\frac{1}{2^i}d^{max}$ (in the unscaled instance), therefore $\frac{1}{2^i}d^{max} < mspan_{OPT}$. We look at the result of the previous run $i-1$: The scaled result was optimal, therefore the unscaled version has a makespan of at most

$$mspan_{ALG} \le mspan_{OPT} + 2n \cdot \varsigma^{i-1} \tag{1}$$

$$= mspan_{OPT} + 2n \cdot \frac{1}{2^{i-1}} \cdot \frac{\varepsilon \cdot d^{max}}{4n} \tag{2}$$

$$= mspan_{OPT} + \varepsilon \cdot \frac{1}{2^i}d^{max} \le (1+\varepsilon)mspan_{OPT}. \tag{3}$$

It should be easy to infer from Lemma 2 that each iteration of this process has polynomial runtime. Combined with the fact that we iterate at most $\log d^{max}$ times we get a runtime that is in $poly(n, \frac{1}{\varepsilon})$. $\qquad\square$

Remark 2 The results of this chapter work, as written, for a constant ψ. Note here, that for series parallel digraphs, this is equivalent to a constant anti-chain size. The algorithms can also be adapted to work on any graph with constant anti-chain size, if the communication delays are bounded by some constant or are *locally small*. Delays are locally small, if for every $(j, k) \in E$, $c(j, k)$ is smaller or equal than every $p_c(k')$, $p_s(k')$, $p_c(j')$ and $p_s(j')$, where k' is every direct successor of j and j' every direct predecessor of k [21].

5 Strong NP-Hardness

In this section, we consider more involved reductions then in Sect. 2 in order to gain a better understanding for the complexity of the problem. First, we show that a classical result due to Lenstra and Rinnooy Kan [12] can be adapted to prove that already the variant of SCS without communication delays and processing times equal to one or two is NP-hard. This already implies strong NP-hardness. Remember that we did show in Sect. 2 that SCS without communication delays and with unit processing times can be solved in polynomial time. Hence, it seems natural to consider the problem variant

with unit processing times and communication delays. We prove this problem to be NP-hard as well via an intricate reduction from $3SAT$ that can be considered the main result of this section. Lastly, we show that the latter reduction can be easily modified to get a strong inapproximability result regarding the general variant of SCS and the cost objective.

5.1 No Delays and Two Sizes

We show strong hardness for the case without communication delays and $p_c(j)$, $p_s(j)$ $\in \{1, 2\}$ for each job j. The reduction is based on a classical result due to Lenstra and Rinnooy Kan [12].

Let $G = (V, E)$, k be a clique instance with $\mid E \mid > \binom{k}{2}$, and let $n = \mid V \mid$ and $m = \mid E \mid$. We construct an instance of the cloud server problem in which the communication delays all equal zero and both the deadline and the cost bound is $2n + 3m$. There is one vertex job $J(v)$ for each node $v \in V$ and one edge job $J(e)$ for each edge $e \in E$ and $J(\{u, v\})$ is preceded by $J(u)$ and $J(v)$. The vertex jobs have size 1 and the edge jobs size 2 both on the server and on the cloud.

Furthermore there is a dummy structure. First, there is a chain of $2n + 3m$ many jobs called the anchor chain. The i-th job of the anchor chain is denoted $A(i)$ for each $i \in \{0, \ldots 2n + 3m - 1\}$ and has size 1 on the cloud and size 2 on the server. Next, there are gap jobs each of which has size 1 both on the server and the cloud. Let $k^* = \binom{k}{2}$ and $v \prec w$ indicate that an edge from v to w is included in the task graph. There are four types of gap jobs, namely $G(1, i)$ for $i \in \{0, \ldots k - 1\}$ with edges $A(2i) \prec G(1, i) \prec A(2(i+1))$, $G(2, i)$ for $i \in \{0, \ldots k^* - 1\}$ with $A(2k+3i+1) \prec G(2, i) \prec A(2k+3(i+1))$, $G(3, i)$ for $i \in \{0, \ldots (n-k)-1\}$ with $A(2k+3k^*+2i) \prec G(3, i) \prec A(2k + 3k^* + 2(i + 1))$, and $G(4, i)$ for $i \in \{0, \ldots (m - k^*) - 1\}$ with $A(2n + 3k^* + 3i + 1) \prec G(4, i) \prec A(2n + 3k^* + 3(i + 1))$ for $i < (m - k^*) - 1$ and $A(2n + 3m - 2) \prec G(4, (m - k^*) - 1)$. Lastly, there are the source and the sink which precedes or succeeds all of the above jobs, respectively.

Lemma 3 *There is a k-clique, if and only if there is a schedule with length and cost at most $2n + 3m$.*

Proof First note that in a schedule with deadline $2n + 3m + 1$ the anchor chain has to be scheduled completely on the cloud. If the schedule additionally satisfies the cost bound, all the other jobs have to be scheduled on the server. Furthermore, for the gap and anchor chain jobs there is only one possible time slot due to the deadline. In particular, $A(i)$ starts at time i, $G(1, i)$ at time $2i + 1$, $G(2, i)$ at time $2k + 3i + 2$, $G(3, i)$ at time $2k + 3k^* + 2i + 1$, and $G(4, i)$ at time $2n + 3k^* + 3i + 2$. Hence, there are k length 1 slots positioned directly before the $G(1, i)$ jobs left on the server, as well as, k^* length 2 slots directly before the $G(2, i)$ jobs, $n - k$ length 1 slots directly before the $G(3, i)$ jobs, and $m - k^*$ length 2 slots directly before the $G(2, i)$ jobs (see also Sect. 4). The m edge jobs have to be scheduled in the length 2 slots, and hence the vertex jobs have to be scheduled in the length 1 slots (Fig. 4).

$\implies$: Given a k-clique, we can position the k clique vertices in the first k length 1 slots, the corresponding k^* edges in the first length 2 slots, the remaining vertex jobs
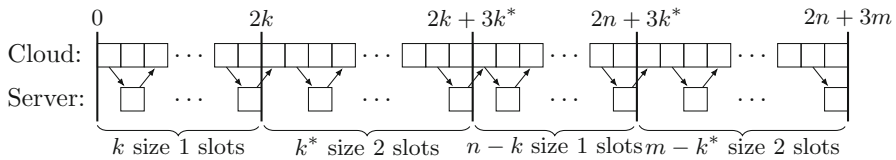
Fig. 4 The dummy structure for the reduction from the clique problem to a special case of SCS. Time flows from left to right, the anchor chain jobs are positioned on the cloud, and the gap jobs on the server

in the remaining length 1 slots, and the remaining edge jobs in the remaining length 2 slots.

$\Longleftarrow$: Given a feasible schedule, the vertices corresponding to the first length 1 slots have to form a clique. This is the case, because there have to be k^* edge jobs in the first length 2 slots and all of their predecessors are positioned in the first length 1 slots. This is only possible if these edges are the edges of a k-clique. $\quad\square$

Hence, we have:

Theorem 10 *The SCS problem with job sizes 1 and 2 and without communication delays is strongly NP-hard.*

In the above reduction the server and the cloud machines are unrelated relative to each other due to different sizes of the anchor chain jobs. However, it is easy to see that the reduction can be modified to a uniform setting where the cloud machines have speed 2 and the server speed 1. If we allow communication delays, even identical machines can be achieved.

5.2 Unit Size and Unit Delay

We consider a unit time variant of our model in which all $p_c = p_s = 1$ and all $c = 1$. Note here, that this also implies that the server and the cloud are identical machines (the cloud still produces costs, while the server does not). As usual for reductions we look at the decision variant of the problem: Is there a schedule with cost smaller or equal to b while adhering to the deadline d.

Theorem 11 *The SCS^1 problem is strongly NP-hard.*

We give a reduction $3SAT \leq_p SCS^1$. Let ϕ be any boolean formula in 3-CNF, denote the variables in ϕ by $\mathcal{X} = \{x_1, x_2, \ldots, x_m\}$ an the clauses by $\mathcal{C} = \{C_1^\phi, C_2^\phi, \ldots, C_n^\phi\}$. Before we define the reduction formula we want to give an intuition and a few core ideas used in the reduction.

The main idea is that we ensure that nearly everything has to be processed on the cloud, there are only a few select jobs that can be handled by the server. For each variable there will be two jobs, of which one can be processed on the server, the selection will represent an assignment. For each clause there will be a job per literal in that clause, only one of which can be processed on the server, and only if the respective variable job is 'true'. Only if for each variable and for each clause one job is handled by the server the schedule will adhere to both the cost and the time limits.
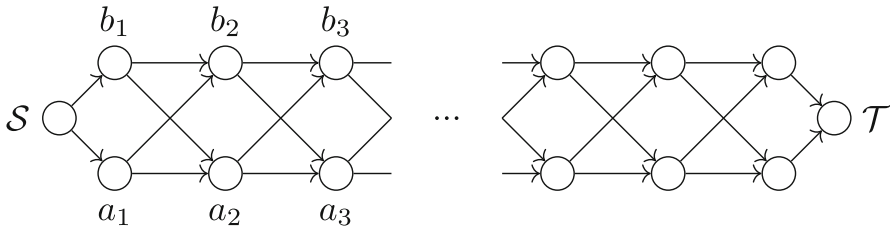
Fig. 5 Schematic representation of an anchor chain

A core technique of the reduction is the usage of an anchor chain. An anchor chain of length l consists of two chains of the same length $l := d - 2$, where we interlock the chains by inserting (a_i, b_{i+1}) and (b_i, a_{i+1}) for two parallel edges (a_i, a_{i+1}) and (b_i, b_{i+1}). The source $\mathcal{S}$ is connected to the two start nodes of the anchor chain, the two nodes at the end of the chain are connected to $\mathcal{T}$ (Fig. 5).

Lemma 4 *If the task graph of a SCS^1 problem contains an anchor chain, every valid schedule has to schedule all but one of a_1, b_1 and one of a_l, b_l on the cloud. For every job a_i, b_i $1 < i < l$ the time step in which it will finish processing on the cloud in every valid schedule is $i + 1$.*

Finally we give the reduction function $f(\phi) = G, d, b$, where $G = (\mathcal{J}, E)$. Set $d = 12 + m + n$ and $b = | \mathcal{J} | - (2 + m + n)$. We define G by constructively giving which jobs and edges are created by f. Create an anchor chain of length $d - 2$, this will be used to limit parts of a schedule to certain time frames. Note that by Lemma 4 we know that every valid schedule of $G = (\mathcal{J}, E), d, \mathbf{k}$ has every node pair of the anchor chain (besides the first and last) on the cloud at a specific fixed timestamp. More specifically, the completion time of a_i and a_{i+j} differ by exactly j time units. For each variable $x_i \in \mathcal{X}$ create two jobs j_{x_i} and $j_{\bar{x}_i}$ and edges (a_{1+i}, j_{x_i}), $(a_{1+i}, j_{\bar{x}_i})$ and (j_{x_i}, a_{5+i}), $(j_{\bar{x}_i}, a_{5+i})$. For each clause C_p^ϕ create a clause job $j_{C_p^\phi}$ and edges $(a_{7+m+p}, j_{C_p^\phi})$ and $(j_{C_p^\phi}, a_{9+m+p})$. Let L_1^p, L_2^p, L_3^p be the literals in C_p^ϕ. Create jobs $j_{L_1^p}, j_{L_2^p}, j_{L_3^p}$ and edges $(j_{L_1^p}, C_p^\phi), (j_{L_2^p}, C_p^\phi), (j_{L_3^p}, C_p^\phi)$ for these literals. For every literal job $j_{L_1^p}$ connect it to the corresponding variable job j_{x_i} or $j_{\bar{x}_i}$ by a chain of length $1 + (m - i) + p$. Also create an edge from a_{3+i} to the start of the created chain and an edge from the end of the chain to a_{6+m+p} (Fig. 6).

It remains to show that there is a schedule of length at most d with costs at most b in $f(\phi) = G, d, b$ if and only if there is a satisfying assignment for ϕ.

Lemma 5 *In a deadline adhering schedule for $f(\phi) = G, d, b$ every job in the anchor chain (except on at the front and one at the end), every job in the variable and clause literal connecting chains and every clause job has to be scheduled on the cloud.*

Proof By Lemma 4 we already know that every node in the anchor chain except one of v_1, w_1 and one of v_l, w_l has to be scheduled on the cloud. We also know, that the jobs in the anchor chain have fixed time steps in which they have to be processed. We look at some chain and its connection to the anchor chain. The start of the chain of
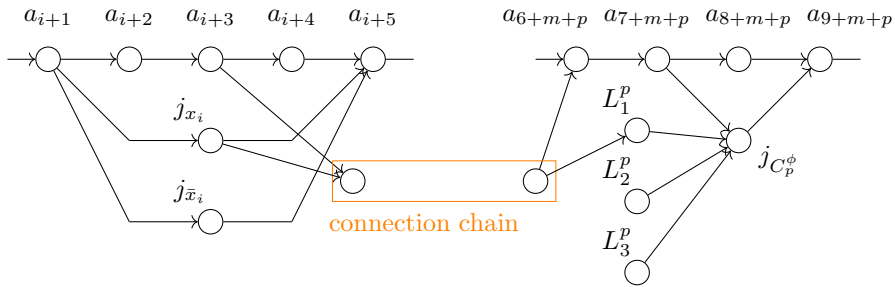
Fig. 6 Schematic representation of the variable and clause gadgets and their connection

length $1 + (m - i) + p$ is connected to a_{3+i}, the end to a_{6+m+p}. Between the end of a_{3+i} and the start of a_{6+m+p} are $6 + m + p - 1 - (3 + i) = 2 + m + p - i$ time steps. So with the processing time required to schedule all $1 + (m - i) + p$ jobs of the chain, there is only one free time step, but we would need at least 2 free time steps to cover the communication cost to and from the server. (Recall here that both a_{3+i} and a_{6+m+p} have to be processed on the cloud). The same simple argument fixes each clause job to a specific time step on the server. □

Lemma 6 *In a deadline adhering schedule for $f(\phi) = G, d, b$ only one of j_{x_i} and $j_{\bar{x}_i}$ can be processed on the server for every variable $x_i \in \mathcal{X}$. The same is true for $j_{L_1^p}, j_{L_2^p}, j_{L_3^p}$ of clause C_p^ϕ.*

Proof j_{x_i} and $j_{\bar{x}_i}$ are both fixed to the same time interval via the edges (a_{1+i}, j_{x_i}), $(a_{1+i}, j_{\bar{x}_i})$ and (j_{x_i}, a_{5+i}), $(j_{\bar{x}_i}, a_{5+i})$. Since a_{1+i} and a_{5+i} will be processed on the cloud and keeping communication delays in mind, only the middle of the three time steps in between can be used to schedule j_{x_i} or $j_{\bar{x}_i}$ on the server. Since the server is only a single machine only on of them can be processed on the server. Note here that the other job can be scheduled a time step earlier which we will later use. The argument for $j_{L_1^p}, j_{L_2^p}, j_{L_3^p}$ works analogously to the statement above.

□

Lemma 7 *There is a deadline adhering schedule for $f(\phi) = G, d, b$ with costs of $| \mathcal{J} | -(2 + m + n)$ if and only if there is a satisfying assignment for ϕ. The variable jobs processed on the cloud represent this satisfying assignment*

Proof From Lemmas 4, 5 and 6 we can infer that a schedule with costs of $| \mathcal{J} | -(2 + m + n)$ has two jobs of the anchor chain, one job for each pair of variable jobs and one job per clause on the server. Two jobs of the anchor chain can always be placed on the server, the choice of variable jobs is also free. It remains to show, that we can only schedule a literal job per clause on the server if and only if the respective clause is fulfilled by the assignment inferred by the variable jobs.

The clause job $j_{C_p^\phi}$ of C_p^ϕ has to be processed in time step $9 + m + p$ (between a_{7+m+p} and a_{9+m+p}). Therefore, $j_{L_1^p}$ has to be processed no later than $8 + m + p$ or $7 + m + p$ if it is processed on the cloud or server respectively. Let j_{x_i} be the variable job connected to $j_{L_1^p}$ via a connection chain.

If j_{x_i} is *true* (scheduled on the cloud), it can finish processing at time step $3+i$, which does not delay the start of the connection chain (which is connected to a_{3+i}, finishing in time step $4+i$). This means that the chain can finish in time step $4+i+1+(m-i)+p = 5+m+p$, the time step $6+m+p$ can be used for communication, allowing $j_{L_1^p}$ to be processed by the server in $7+m+p$.

If j_{x_i} is *false* (scheduled on the server), it finishes processing at time step $4+i$, which, combined with the induced communication delay, delays the start of the chain by 1. Therefore, the chain only finishes in time step $6+m+p$, and $j_{L_1^p}$ has to be processed on the cloud, since there is not enough time for the communication back and forth.

Trivially, the same argument holds true for $j_{L_2^p}$ and $j_{L_3^p}$.

□

It should be easy to see that the reduction function f is computable in polynomial time. Combined with Lemma 7 this concludes the proof of our reduction $3SAT \leq_p SCS^1$. The correctness of Theorem 11 trivially follows from that.

5.2.1 The General Case

Adapting the previous reduction we can show an even stronger result for the general case of SCS. Basically we are able to degenerate the reduction output in a way, that a satisfying assignment results in a schedule with cost 0, while every other assignment (schedule) has costs of at least 1. It should be obvious, that this also means that there is no approximation algorithm for this problem with a fixed multiplicative performance guarantee, if $P \neq NP$.

This reduction uses processing times and communication delays of $0, \infty$ and values in between. Note that ∞ can simply be replaced by $d+1$. To keep the following part readable we again substitute "an edge (j, j') with communication delay $c(j, j') = k$" simply by "an edge $c(j, j') = k$"

We follow the same general structure (an anchor chain, variable-, clause- and connection gadgets). The *anchor chain* now looks as follows: For every time step create two jobs a_i and a_i' with $p_s(a_i) = 0$, $p_c(a_i) = \infty$, $p_s(a_i') = \infty$, $p_c(a_i') = 0$ and an edge $c(a_i, a_i') = 0$. These chain links are than connected by an edge $c(a_i', a_{i+1}) = 1$. Finally we create $c(S, a_1) = 1$ and $c(a_d, T) = 0$. It should be easy to see, that every schedule will process a_i and a_i' in time step i on the server and the cloud respectively. This gives us anchors to the server and to the cloud for every time step, without inducing congestion or costs. Since the anchor jobs themselves have processing time of 0, the "usable" time interval between some a_i and a_{i+1} is one full time step.

For each variable $x_i \in \mathcal{X}$ create two jobs $j_{x_i}, j_{\bar{x}_i}$ with $p_s(j_{x_i}) = p_s(j_{\bar{x}_i}) = 1$ and $p_c(j_{x_i}) = p_c(j_{\bar{x}_i}) = 0$. Create edges $c(a_i, j_{x_i}) = 1$, $c(a_i, j_{\bar{x}_i}) = 1$ and $c(j_{x_i}, a_{i+1}) = 0$, $c(j_{\bar{x}_i}, a_{i+1}) = 0$. In short, only one of them can be processed on the server, the other on the cloud. Both will finish in time step $i+1$, the one processed on the server is *true*, therefore processing both on the cloud is possible, but not helpful.

For each clause C_p^ϕ create a clause job $j_{C_p^\phi}$ with $p_s(j_{C_p^\phi}) = \infty$, $p_c(j_{C_p^\phi}) = 0$ and edges $c(a_{5+m+3p}', j_{C_p^\phi}) = \infty$ and $c(j_{C_p^\phi}, a_{6+m+3p}') = \infty$. This means, that $j_{C_p^\phi}$

has to finish processing by time step $6 + m + 3p$. Let L_1^p, L_2^p, L_3^p be the literals in C_p^ϕ. Create jobs $j_{L_1^p}, j_{L_2^p}, j_{L_3^p}$ each with $p_c = p_s = 1$ and edges $c(j_{L_1^p}, C_p^\phi) = 0$, $c(j_{L_2^p}, C_p^\phi) = 0, c(j_{L_3^p}, C_p^\phi) = 0$ for these literals. Create edges $c(a_3 + m + 3p, j_{L_1^p}) = 0, c(a_3 + m + 3p, j_{L_2^p}) = 0$ and $c(a_3 + m + 3p, j_{L_3^p}) = 0$, so that, in theory, all three of the literal jobs can be processed on the server, finishing in time steps $4 + m + 3p$, $5 + m + 3p$ and $6 + m + 3p$ respectively. Lastly, for every literal job $j_{L_1^p}$ connect it to the corresponding variable job j_{x_i} (or $j_{\bar{x}_i}$) by a an edge with communication delay of $m - i + 3p + 3$. Since j_{x_i} (or $j_{\bar{x}_i}$) finish processing in time step $i + 1$, this means that $j_{L_1^p}$ can start no earlier than $m + 3p + 4$ (and therefore finish processing in $5 + m + 3p$), if j_{x_i} (or $j_{\bar{x}_i}$) were processed on the cloud.

Recall here, that a variable job being scheduled on the server denotes that it is *true*. So only a literal job that evaluates to true, can be scheduled so that it finishes processing in time step $4 + m + 3p$ on the cloud.

It follows directly, that a schedule for this construction will have costs of 0 if and only if the assignment derived from the placement of the variable jobs fulfills every clause.

Theorem 12 *There is no approximation algorithm for SCS that has a fixed performance guarantee, assuming that $P \neq NP$.*

6 Unit Size and Unit Delay—And No Delay

As the last step of this paper we explore simple algorithms on unit size instances with arbitrary task graphs. Recall that we proved these to be strongly NP-hard. We use resource augmentation and ask: given a SCS^1 problem instance with deadline d, find a schedule in poly. time that has a makespan of at most $(1 + \varepsilon) \cdot d$ that *approximates* the optimal cost in regards to the actual deadline d.

If there is a chain of length d or $d - 1$, that chain has to be scheduled on the server, since there is no time for the communication delay. For instances with a chain of size d that is trivially optimal, for those with $d - 1$ we can check in polynomial time if any other job also fits on the server, again, finding an optimal solution. From now we assume that there is no chain of length more than $d - 2$.

First, construct a schedule which places every job on the cloud, as fast as possible. The resulting schedule from time step (ts) 1 to $(1 + \varepsilon) \cdot d$ looks as follows: one ts of communication, at most $d - 2$ ts of processing on the server, another ts for communication followed by at least εd empty ts. Now pull (one of) the last job(s) that is processed on the cloud to the last empty ts and process it on the server instead. Repeat this process until the last job can not be moved to the server anymore. Do the whole procedure again, but this time starting with the cloud schedule in the end of the schedule, and each time pulling the first job to the beginning. Keep the result with lower costs. Note that one can always fill the ts being used solely for communicating from the server to the cloud with processing one job on the server, that otherwise would be one of the first jobs being processed on the cloud (the same holds for the other direction).

Theorem 13 *The described algorithm yields a schedule with approximation factor of* $\frac{1+\varepsilon}{2\varepsilon}$ *while having a makespan of at most* $(1+\varepsilon)\cdot d$.

Proof Case $n \leq (1+\varepsilon)d$: The algorithm places all jobs on the server, the cost is 0 and therefore optimal.

Case $(1+\varepsilon)d < n < (1+2\varepsilon)d$: Assume that the preliminary cloud-only schedule needs $d-2$ ts on the cloud, if that is not the case, we *stretch* the schedule to that length. There are n jobs distributed onto $d-2$ ts. Therefore, either from the front or from the end, there is an interval of length $\frac{d}{2}-1$ with at least $\frac{d}{2}-1$ and at most $\frac{n}{2} < \frac{(1+2\varepsilon)d}{2} = \frac{d}{2} + \varepsilon d$ many jobs. It should be easy to see, that the algorithm will schedule those at most $\frac{d}{2} + \varepsilon d - 1$ jobs to the $\frac{d}{2} - 1$ plus the free εd many time slots. If the interval included less than $\frac{d}{2} + \varepsilon d - 1$ jobs, it will simply continue until the $\frac{d}{2} - 1 + \varepsilon d$ ts are filled with jobs being processed on the server. With the one job we can process on the server during the communication ts we process $\frac{d}{2} + \varepsilon d$ jobs on the server and have costs of $n - (\frac{d}{2} + \varepsilon d)$. An optimal solution has costs of at least $n - d$. For $\varepsilon \geq 0.5$ it holds that: $cost_{ALG} = n - (\frac{d}{2} + \varepsilon d) \leq n - d \leq cost_{OPT}$, otherwise:

$$\frac{cost_{ALG}}{cost_{OPT}} \leq \frac{n - (\frac{d}{2} + \varepsilon d)}{n - d} \leq \frac{(1+\varepsilon)d - (\frac{d}{2} + \varepsilon d)}{(1+\varepsilon)d - d} \leq \frac{0.5d}{\varepsilon d} = \frac{1}{2\varepsilon}$$

Case $(1+2\varepsilon)d \leq n$: In this case we simply observe that our algorithm places at least εd many jobs on the server. For $\varepsilon \geq 1$ it holds that: $cost_{ALG} = n - \varepsilon d \leq n - d \leq cost_{OPT}$, otherwise:

$$\frac{cost_{ALG}}{cost_{OPT}} \leq \frac{n - \varepsilon d}{n - d} \leq \frac{(1+2\varepsilon)d - \varepsilon d}{(1+2\varepsilon)d - d} = \frac{d + \varepsilon d}{2\varepsilon d} = \frac{1 + \varepsilon}{2\varepsilon}$$

□

6.1 No Delays and Identical Machines

We design a simple heuristic for the case in which the server and the cloud machines behave the same, that is, $p_c(j) = p_s(j)$ for each job j (except for the source and sink), and the communication delays all equal zero. In this case, we may define the length of a chain in the task graph as the sum of the processing times of the jobs in the chain. The first step in the algorithm is to identify a longest chain in the task graph, which can be done in polynomial time. The jobs of the longest chain are scheduled on the server and the remaining jobs on the cloud each as early as possible. Now, the makespan of the resulting schedule is the length of a longest chain, which is optimal (or better) and there are no idle times on the server. However, the schedule may not be feasible since the budget may be exceeded. Hence, we repeatedly do the following: If the budget is still exceeded, we pick a job scheduled on the cloud with maximal starting time and move it on to the server right before its first successor (which may be the sink). Some jobs on the server may be delayed by this but we can do so without causing idle times. If all the processing times are equal this procedure produces an

optimal solution and otherwise there may be an additive error of up to the maximal job size. Hence, we have:

Theorem 14 *There is a* 2-*approximation for SCS without communication delays and identical server and cloud machines.*

It is easy to see, that the analysis is tight considering an instance with three jobs: One with size b, one with size $b + \varepsilon$, and one with size 2ε. The first jobs precedes the last one. Our algorithm will place everything on the server, while the first job is placed on the cloud in the optimal solution.

Note that we can take a similar approach to find a solution with respect to the cost objective by placing more and more jobs on the server as long as the deadline is still adhered to. However, an error of one job can result in an unbounded multiplicative error in the objective in this case. On the other hand, it is easy to see that in the case with unit processing times, there will be no error at all in both procedures yielding:

Corollary 2 *The variant of SCS without communication delays and unit processing times can be solved in polynomial time with respect to both the makespan and the cost objective.*

7 Generalizations of Server Cloud Scheduling

In this chapter we introduce some generalizations to the SCS. We consider different aspects from multiple clouds and server machines to direction specific delays. We sketch how to adapt our algorithms for SCS^e and SCS^ψ to cover those new generalizations.

7.1 Changes in the Definitions

We shortly define the changes to the model that we explore in this section.

7.1.1 Machine Model

So far we imagined a single server machine and one homogeneous cloud in our problem definition. Now, instead of a single server machine there can be any (constant) number of identical server machines: SERVER $= \{s_1, \ldots, s_z\}$. Instead of one homogeneous cloud there can be any number of different cloud contexts: CLOUDS $= \{c_1, \ldots, c_k\}$. Each cloud context still consists of an unlimited number of parallel machines.

7.1.2 Jobs

Jobs are still given as a task graph $G = (\mathcal{J}, E)$. A job $j \in \mathcal{J}$ has processing time $p_s(j)$ on any server machine and processing time $p_{c_i}(j)$ on a machine of cloud context c_i. An edge $e = (i, j)$ and machine contexts $m_1, m_2 \in \{s, c_1, \ldots, c_k\}$ have a communication delay of $c_{m_1 \triangleright m_2}(i, j) \in \mathbb{N}_0$, which means, that after job i finished on a machine of type m_1, j has to wait an additional $c_{m_1 \triangleright m_2}(i, j)$ time steps before it can start on a

machine of type m_2. For $m_1 = m_2$ we set $c_{m_1 \triangleright m_2}(i, j) = 0$. Note that this function does not need to be symmetric, e.g. $c_{m_1 \triangleright m_2}(i, j)$ and $c_{m_2 \triangleright m_1}(i, j)$ may be unequal.

7.1.3 Costs and Schedules

Previously we defined cost simply by "time spend on the cloud". While considering multiple clouds, that is not sensible anymore. A faster cloud will not be universally cheaper than a slower one. We define a cost function based on the cloud context and job, $cost : \mathcal{J} \times \text{CLOUDS} \mapsto \mathbb{N}_0$. A schedule still consists of $C : \mathcal{J} \mapsto \mathbb{N}_0$ (maps jobs to their completion time), but instead of a partition we give a mapping function $\eta : \mathcal{J} \mapsto \{s_1, \ldots, s_z\} \cup \{c_1, \ldots, c_k\}$. Note that s_i refers to one specific server machine, while c_i refers to a cloud context, consisting of infinitely many machines.

We call a schedule $\pi = (C, \eta)$ valid if and only if the following conditions are met:

(a) There is always at most one job processing on each server:

$$\forall_{i,j \in \mathcal{J}, i \neq j : \eta(i) = \eta(j) \in \text{SERVER}} : (C(i) \leq C(j) - p_s(j)) \vee (C(i) - p_s(i) \geq C(j))$$

(b) Tasks are not started before the previous tasks has been finished/ the required communication is done:

$$\forall_{(i,j) \in E} : (C(i) + c_{\eta(i) \triangleright \eta(j)}(i, j) \leq C(j) - p_{\eta(j)} j)$$

The makespan ($mspan$) of a schedule is still given by the completion time of the sink $\mathcal{T}$: $C(\mathcal{T})$. The cost ($cost$) of a schedule is given by:

$$\sum_{j \in jobs : \eta(j) \in clouds} cost(j, \eta(j)).$$

7.2 Revisiting SCS^e

We briefly sketch how to adapt the algorithm from Sect. 3 to incorporate the previously defined changes on the model. We will use the observations, that multiple server machines only affect the scheduling of parallel parts and that we can always calculate an optimal cloud location for a job in a given situation (part of the schedule, time frame and location of predecessor and successor).

Theorem 15 *There is a $(4 + \varepsilon)$-approximation algorithm for the budget restrained makespan minimization problem on extended chains, even when there are z server machines, k different cloud contexts, the communication delays are directionally dependent on the machine context, and costs are given as an arbitrary cost function* $cost : \mathcal{J} \times \text{CLOUDS} \mapsto \mathbb{N}_0$.

Proof We adapt the pseudo polynomial algorithm from Sect. 3 that given a feasible makespan estimate $T (T \geq mspan_{OPT})$ calculates a schedule with makespan of at most $\min\{2T, 2mspan_{OPT}\}$, such that it incorporates the changes to the model and

calculates a schedule with makespan of at most $\min\{4T + \varepsilon', 4mspan_{OPT} + \varepsilon'\}$. The only change in the state description is that $loc \in \{s, c_1, \ldots, c_k\}$ instead of $loc \in \{s, c\}$. As the state description is used for the chain parts of the extended chain, we do not differentiate the server machines here. The creation of the state extension list EXTENSIONS^j (each of form $[\Delta t, loc_{j-1} \rightarrow loc_j] = cost$), has the following changes:

- Instead of the four combinations $s \rightarrow s, s \rightarrow c, c \rightarrow s, c \rightarrow c$, we consider all combinations from $\{s, c_1, \ldots, c_k\} \times \{s, c_1, \ldots, c_k\}$.
- Substitute the corresponding values, for example $[p_c(j) + c(j - 1, j), \ s \rightarrow c] = p_c(j)$ becomes $[p_{c_i}(j) + c_{s \triangleright c_i}(j - 1, j), \ s \rightarrow c_i] = cost(j, c_i)$.
- If there is a parallel subgraph between $j - 1$ and j we adapt the calculation in the following way:
  - Calculate Δ^{max} as before (the sum over all processing times on the server plus the biggest relevant in- and outgoing communication delays)
  - Iterate over Δ^i in $\{0, \ldots, \Delta^{max}\}$:
    * As before, check for each job if it fits: (1) only on the servers, (2) not on the servers but on at least one cloud context, (3) on both, (4) on none. If at least one job falls into (4) break.
    * Calculate for each job j in (2) or (3) the cheapest fitting option to schedule that job on some available cloud in time frame Δ^i. Use that cost c_j for j for the remainder of the iteration.
    * Greedily put jobs in (1) onto server machines (1 to k) until the current server has load $\geq \Delta^i$, proceed with the next machine and so on. If not all jobs in (1) can be placed this way break, as there is not enough space to place jobs on the server that do not fit on the cloud in the given time frame.
    * Sort the jobs in (3) by their ratio of cost c_j to processing time on the server (highest to lowest cost per time). Continue by greedily placing those on the server machines as before. When all jobs in (3) are placed, or all server machines have load $\geq \Delta^i$, put all remaing jobs from (3) on their corresponding cheapest cloud context.
    * Put all jobs from (2) on their corresponding cheapest cloud context.
    * insert time in the front and back corresponding to the biggest communication delay invoked by the (sub-)schedule for the parallel part

The rest of the algorithm behaves as before. The changes to state extensions spanning a parallel subgraph calculate solutions that have at most optimal cost for a time frame of Δ^i, while using a time frame of $4\Delta^i$. The 4 times correspond to: at most $2\Delta^i$ time for all in- and outgoing communication delays since the communication delays have to fit into Δ^i to be considered, at most $2\Delta^i$ time for our greedy packing of the server machines since we can add a job of size Δ^i to a machine currently having load $\Delta^i - \epsilon$. It should be easy to see that the greedy packing of "highest cost jobs", with what is essentially resource augmentation of a multiple knapsack problem, gives at most optimal cost. Note that we could also utilize a PTAS for multiple knapsack here to stay in a time frame of $3\Delta^i$, but we want to find a solution with optimal cost (or lower), to remain strictly budget adhering.

It remains to simply use the same scaling technique used in Sect. 3 to get the $4 + \varepsilon$-approximation.

$\square$

If the communication delays are constant the result can be easily adapted to yield a $2 + \varepsilon$-approximation, by getting rid of the added time for communication delays.

7.3 Revisiting *SCS$^\Psi$*

In a similar vein as the previous subsection we briefly sketch how to adapt the results from Sect. 4 to include most of the previously defined model generalizations. Naturally, we still require the *maximum cardinality source and sink dividing cut* to be bounded by a constant. In contrast to the previous result we require the number of server machines to be a constant.

Theorem 16 *There is an FPTAS for the budget restrained makespan minimization problem for graphs with a constant* maximum cardinality source and sink dividing cut, *even when there are a constant number of server machines, k different cloud contexts, the communication delays are directionally dependent on the machine context, and costs are given as an arbitrary cost function cost* $: \mathcal{J} \times \text{CLOUDS} \mapsto \mathbb{N}_0$.

Proof We make the following two changes to the state definition: We consider $loc_j \in \{s, c_1, \ldots, c_k\}$ instead of $loc_j \in \{s, c\}$, we track the unused time of every server machine individually so instead of a single f_s the state contains $f_{s^1}, \ldots, f_{s^z}$. The dynamic program needs only minor tweaks. When iterating through the jobs that are open (and of which all predecessors have been processed) use the server s^i with the smallest fitting f_{s^i} and set $f_{s^i} = 0$. Instead of checking if the job fits on "the cloud" we simply go through all clouds, and add corresponding states for each fitting location. While calculating the value of a state use the new cost function *cost* instead of p_c, while checking if a job fits we use the directional communication delays. After a full iteration increase each f_{s^i} by one (instead of only increasing the singular f_s). It should be easy to see, that these adaptations do not change the correctness of the algorithm. The runtime (after the rounding technique) naturally increases to $poly(n^z, k, \frac{1}{\varepsilon})$, which is polynomial, iff z (the number of server machines) is a constant.

$\square$

8 Approximating the Pareto Front

The problem variants we describe and analyze in this paper are multi-criteria optimization problems. To simultaneously handle the two criteria cost and makespan, we either looked at decision variants "is there a schedule with makespan $\leq d$ and cost $\leq b$" or we used one of them as a constraint and asked "given a budget of b, minimize the makespan" (or vice versa). Naturally, one might be interested in finding an assortment of different efficient solutions, without giving a specific budget or deadline. A solution is called efficient, or Pareto optimal, if we can not improve one of the criteria, without worsening the other. The set of all Pareto optimal solutions is called the Pareto front.
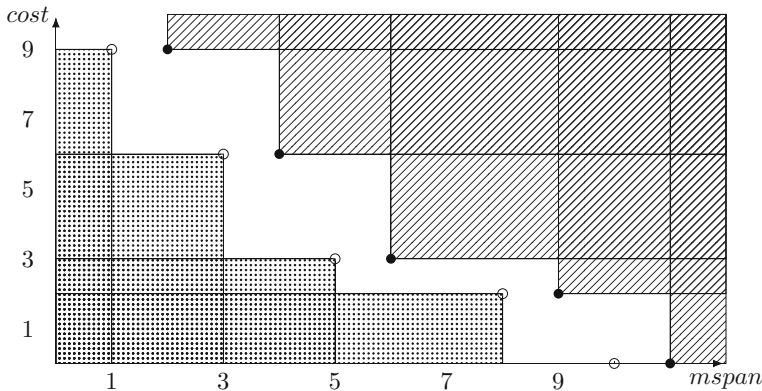
Fig. 7 Reported solutions by our algorithm, filled circles and empty circles represent reported points and best possible solutions due to the approximation factor, respectively. Dotted region is infeasible, striped region is feasible but dominated

In the following, we will use the term *point* to refer to the makespan and cost of a feasible solution of a given SCS problem.

For our NP-hard problems, we will not be able to efficiently calculate the exact Pareto front, but we can find a set of points that is close to the optimum. In the literature, one can find slightly different definitions for such approximations. In [22], the authors scale each criteria to an interval from 0 to 1. A set of points is an α-approximation, if for each point in the actual Pareto front, there is a point where each dimension is offset by at most an additional $\pm\alpha$. We follow the definition of Pareto front approximations given in [23] (adapted to our case with exactly 2 objectives):

Definition 1 A set of points S is an α-approximation of a Pareto front, if for each point $p = (mspan^p, cost^p)$ there is a point $p' = (mspan^{p'}, cost^{p'})$ in S with $mspan^{p'} \leq (1 + \alpha)mspan^p$ and $cost^{p'} \leq (1 + \alpha)cost^p$.

The dynamic programming algorithms established in this paper can be used to find such an approximation. We use the results from Sect. 4 to show how this is done, but note that a similar approach can be used for other results of this paper.

Intuitively our dynamic programs calculate a collection of possible results but only report a single one, where the "best" is selected based on the current objective. Imagine that one of our deadline restrained algorithms with approximation factor $(1+\varepsilon)$ reports *every* non dominated solution it finds instead. The result for $d = 10$ and $\varepsilon = 0.1$ could look like Fig. 7. For every reported point $(mspan, cost)$ we can infer a lower bound on the makespan of $mspan - \varepsilon \cdot d$ any schedule with a given $cost$ has, due to the approximation factor of the algorithm. Note that gap is in relation to a given d, and therefore results with a smaller makespan are less precise. We will circumvent that by repeating the algorithm with smaller values for d.

Theorem 17 *Using* DPFGG *(Algorithm* 1*) one can α-approximate the Pareto front of a SCS problem with constant ψ in polynomial time, for any $\alpha > 0$.*

Proof Given some SCS problem with constant ψ run DPFGG (with the rounding approach) with $d = \sum_{j \in \mathcal{J}} p_s(j)$. Normally the algorithm found the first state $[\hat{d}, f_s] = cost$. Now, instead let the algorithm find the first state $[t, f_s] = cost$ for every $t \in (0.5\hat{d}, \hat{d}]$. For each of those states calculate an upper bound on the makespan for the respective schedule in the unscaled instance. Following the argumentation in the proof for Theorem 8, we know that the makespan is $\leq t + \varsigma + (n-2)2\varsigma = (t+2n-4)\varsigma$. Report the point $(mspan = (t + 2n - 4)\varsigma, cost)$ and add it to S. After that full algorithm iteration, set $d := 0.5d$ and repeat the process. Do this until $d = 1$. Finally, return the reported point set S.

We want to show that for every point $p = (mspan^p, cost^p)$ of a Pareto front, there is a reported point $p' = (mspan^{p'}, cost^{p'})$ with $mspan^{p'} \leq (1 + \alpha)mspan^p$ and $cost^{p'} \leq (1 + \alpha)cost^p$. Given some point $p = (mspan^p, cost^p)$, look at the iteration where $0.5d < mspan^p \leq d$. Since there is a feasible schedule with $mspan^p$ and $cost^p$ at some point during that iteration we found a feasible scaled schedule with $t = \lfloor \frac{mspan^p}{\varsigma} \rfloor$ and $cost \leq cost^p$. The calculated upper bound for that schedule in unscaled is then $(\lfloor \frac{mspan^p}{\varsigma} \rfloor + 2n - 4)\varsigma \leq mspan^p + (2n - 4)\varsigma = mspan^p + (2n - 4)\frac{\varepsilon \cdot d}{2n} \leq mspan^p + \varepsilon d \leq (1 + 2\varepsilon)mspan^p$ (recall: $\varsigma := \frac{\varepsilon \cdot d}{2n}$). Therefore, a point $p' = (mspan^{p'}, cost^{p'})$ with $mspan^{p'} \leq (1 + 2\varepsilon)mspan^p$ and $cost^{p'} \leq cost^p$ got reported. Setting $\varepsilon = 0.5\alpha$ and noting that we repeat the process no more than $log(\sum_{j \in \mathcal{J}} p_s(j))$ times concludes the proof. □

9 Future Work

We give a small overview over the future research directions that emerge from our work. **SCS^e**: If good approximations for $1 \mid r_j \mid \sum w_j U_j$ become established, the algorithm given in Sect. 3 for the extended chain could probably be improved. One could model the incoming communication delay with release dates and get an equivalent subproblem to solve, instead of the approximate subproblem currently used. **SCS**: Sect. 5 gives a strong inapproximability result for the general case with regards to the cost function. For two easy cases (chain and fully parallel graphs) we could establish FPTAS results, for graphs with a constant ψ we have an algorithm that finds optimal solutions with a $(1 + \varepsilon)$ deadline augmentation. Here one could explore if there are FPTAS results for different assumptions, are there approximation algorithms without resource augmentation for constant ψ instances, and lastly are there approximation algorithms with resource augmentation for the general case. For the makespan function we already have a FPTAS for graphs with a constant ψ. It remains to explore approximation algorithms or inapproximability results for the general case of this problem. **SCS^1**: We show strong NP-hardness even for this simplified problem. Since this is a special case of the general problem all constructive results still hold, additionally we were able to give a first simple algorithm for cost optimization in general graphs. Here

it would be interesting to look into more involved approximation algorithms that give better performance guarantees, maybe without resource augmentation.

Declarations

Conflicts of interest The authors are not aware of any conflict of interests.

References

1. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Syst. Tech. J. **45**(9), 1563–1581 (1966). https://doi.org/10.1002/j.1538-7305.1966.tb01709.x
2. Levey, E., Rothvoss, T.: A (1+epsilon)-approximation for makespan scheduling with precedence constraints using LP hierarchies. In: Wichs, D., Mansour, Y. (eds.) Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18–21, 2016, pp. 168–177 (2016). https://doi.org/10.1145/2897518.2897532
3. Garg, S.: Quasi-PTAS for scheduling with precedences using LP hierarchies. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9–13, 2018, Prague, Czech Republic. LIPIcs, vol. 107, pp. 59–15913 (2018). https://doi.org/10.4230/LIPIcs.ICALP.2018.59
4. Kulkarni, J., Li, S., Tarnawski, J., Ye, M.: Hierarchy-based algorithms for minimizing makespan under precedence and communication constraints. In: Chawla, S. (ed.) Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5–8, 2020, pp. 2770–2789 (2020). https://doi.org/10.1137/1.9781611975994.169
5. Davies, S., Kulkarni, J., Rothvoss, T., Tarnawski, J., Zhang, Y.: Scheduling with communication delays via LP hierarchies and clustering. In: 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16–19, 2020, pp. 822–833 (2020). https://doi.org/10.1109/FOCS46700.2020.00081
6. Davies, S., Kulkarni, J., Rothvoss, T., Tarnawski, J., Zhang, Y.: Scheduling with communication delays via LP hierarchies and clustering II: weighted completion times on related machines. In: Marx, D. (ed.) Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021, pp. 2958–2977 (2021). https://doi.org/10.1137/1.9781611976465.176
7. Aba, M.A., Kordon, A.M., Pallez, G.: Scheduling on two unbounded resources with communication costs. In: Yahyapour, R. (ed.) Euro-Par 2019: Parallel Processing—25th International Conference on Parallel and Distributed Computing, Göttingen, Germany, August 26–30, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11725, pp. 117–128 (2019). https://doi.org/10.1007/978-3-030-29400-7_9
8. Saha, B.: Renting a cloud. In: Seth, A., Vishnoi, N.K. (eds.) IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12–14, 2013,

Guwahati, India. LIPIcs, vol. 24, pp. 437–448 (2013). https://doi.org/10.4230/LIPIcs.FSTTCS.2013.437

9. Mäcker, A., Malatyali, M., auf der Heide, F.M., Riechers, S.: Cost-efficient scheduling on machines from the cloud. J. Comb. Optim. **36**(4), 1168–1194 (2018). https://doi.org/10.1007/s10878-017-0198-x

10. Maack, M., auf der Heide, F.M., Pukrop, S.: Server cloud scheduling. In: Könemann, J., Peis, B. (eds.) Approximation and Online Algorithms—19th International Workshop, WAOA 2021, Lisbon, Portugal, September 6–10, 2021, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12982, pp. 144–164 (2021). https://doi.org/10.1007/978-3-030-92702-8_10

11. Lenstra, J.K., Kan, A.R., Brucker, P.: Complexity of machine scheduling problems. Ann. Discrete Math. **1**, 343–362 (1977). https://doi.org/10.1016/S0167-5060(08)70743-X

12. Lenstra, J.K., Kan, A.H.G.R.: Complexity of scheduling under precedence constraints. Oper. Res. **26**(1), 22–35 (1978). https://doi.org/10.1287/opre.26.1.22

13. Svensson, O.: Hardness of precedence constrained scheduling on identical machines. SIAM J. Comput. **40**(5), 1258–1274 (2011). https://doi.org/10.1137/100810502

14. Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.R.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discrete Math. **5**, 287–326 (1979). https://doi.org/10.1016/S0167-5060(08)70356-X

15. Lawler, E.L., Moore, J.M.: A functional equation and its application to resource allocation and sequencing problems. Manag. Sci. **16**(1), 77–84 (1969). https://doi.org/10.1287/mnsc.16.1.77

16. Sahni, S.: Algorithms for scheduling independent tasks. J. ACM **23**(1), 116–127 (1976). https://doi.org/10.1145/321921.321934

17. Adamu, M.O., Adewumi, A.O.: A survey of single machine scheduling to minimize weighted number of tardy jobs. J. Ind. Manag. Optim. **10**(1), 219 (2014). https://doi.org/10.3934/jimo.2014.10.219

18. Detienne, B.: A mixed integer linear programming approach to minimize the number of late jobs with and without machine availability constraints. Eur. J. Oper. Res. **235**(3), 540–552 (2014). https://doi.org/10.1016/j.ejor.2013.10.052

19. Sevaux, M., Dauzère-Pérès, S.: Genetic algorithms to minimize the weighted number of late jobs on a single machine. Eur. J. Oper. Res. **151**(2), 296–306 (2003). https://doi.org/10.1016/S0377-2217(02)00827-5

20. M'Hallah, R., Bulfin, R.L.: Minimizing the weighted number of tardy jobs on a single machine with release dates. Eur. J. Oper. Res. **176**(2), 727–744 (2007). https://doi.org/10.1016/j.ejor.2005.08.013

21. Möhring, R.H., Schäffter, M.W., Schulz, A.S.: Scheduling jobs with communication delays: Using infeasible solutions for approximation (extended abstract). In: Díaz, J., Serna, M.J. (eds.) Algorithms—ESA '96, Fourth Annual European Symposium, Barcelona, Spain, September 25–27, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1136, pp. 76–90 (1996). https://doi.org/10.1007/3-540-61680-2_48

22. Legriel, J., Guernic, C.L., Cotton, S., Maler, O.: Approximating the pareto front of multi-criteria optimization problems. In: Esparza, J., Majumdar, R. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20–28, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6015, pp. 69–83 (2010). https://doi.org/10.1007/978-3-642-12002-2_6

23. Papadimitriou, C.H., Yannakakis, M.: On the approximability of trade-offs and optimal access of web sources. In: 41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12–14 November 2000, Redondo Beach, California, USA, pp. 86–92 (2000). https://doi.org/10.1109/SFCS.2000.892068