European Journal of Operational Research 291 (2021) 629–639

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Single-machine scheduling with release times, deadlines, setup times, and rejection

Mathijs de Weerdt[a,1,*], Robert Baart[a,1], Lei He[a,b,1]

[a] Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Van Mourik Broekmanweg 6, XE Delft 2628, the Netherlands
[b] College of Systems Engineering, National University of Defense Technology Changsha 410073, China

ARTICLE INFO

Article history:
Received 5 October 2019
Accepted 28 September 2020
Available online 3 October 2020

Keywords:
Scheduling
Order acceptance
Dynamic programming
Decision diagrams
Fixed-parameter tractability

ABSTRACT

Single-machine scheduling where jobs have a penalty for being late or for being rejected altogether is an important (sub)problem in manufacturing, logistics, and satellite scheduling. It is known to be NP-hard in the strong sense, and there is no polynomial-time algorithm that can guarantee a constant-factor approximation (unless P=NP). We provide an exact algorithm that is fixed-parameter tractable in the slack and the maximum number of time windows overlapping at any point in time, i.e., the *width*. This algorithm has a runtime exponential in these parameters, but quadratic in the number of jobs, even when modeling sequence-dependent setup times. We further provide a fixed-parameter fully-polynomial time approximation scheme (FPTAS) with only this width as a parameter, having a runtime bound that is cubic. Finally, we propose a neighbourhood heuristic similar to the Balas-Simonetti neighbourhood. All algorithms use an efficient representation of the state space inspired by decision diagrams, where partial solutions that are provably dominated are excluded from further consideration. Experimental evidence shows that the exact method significantly outperforms the state-of-the-art on instances where the width is smaller than one third of the number of jobs and finds optimal solutions to previously unsolved instances. The FPTAS is competitive to state-of-the-art heuristics only when the width is significantly smaller, but the neighbourhood heuristic outperforms most other heuristics in runtime or quality.

1. Introduction

Satellite observation scheduling (Bianchessi, Cordeau, Desrosiers, Laporte, & Raymond, 2007), order acceptance and scheduling in make-to-order systems (Oguz, Sibel Salman, & Bilgintürk Yalçın, 2010), and the orienteering problem with time windows (Gunawan, Lau, & Vansteenwegen, 2016; Labadie, Mansini, Melechovský, & Wolfler Calvo, 2012) can all be seen as instances of a single-machine order acceptance and scheduling problem with sequence-dependent setup times (Slotnick, 2011). This problem involves both a decision on including an order/visit and finding a schedule/sequence that meets release time and (hard) deadline constraints, may need to account for setup/travel times that depend on the previous order in the sequence, and simultaneously minimizes the total tardiness. In the remainder of this paper, we refer to this scheduling problem with release times, deadlines and rejection as the Order Acceptance and Scheduling (OAS) problem. OAS is at least as hard as single machine with release times and deadlines (without the "acceptance" and sequence-dependence elements), which is strongly NP-hard by a trivial reduction from $1|r_i|L_{max}$ (Brucker, 2007), even if instances are restricted to contain processing times of only two different non-unit lengths (Elffers & de Weerdt, 2017).

1.1. Problem definition

In the order acceptance and scheduling problem the aim is to maximize the revenue of accepted jobs minus their tardiness penalty. First, however, we express this (as a minimization) problem in the commonly used three-field notation for scheduling problems (Pinedo, 2012):

$$1 \mid r_j; s_{jk}; \text{reject}; \bar{d}_j \mid \sum_{j \notin R} w_j T_j - \sum_{j \notin R} v_j.$$

In this notation, the 1 is for the single machine, r_j means that every job j has a release time r_j. The term s_{jk} means that jobs

* Corresponding author.
E-mail address: M.M.deWeerdt@tudelft.nl (M. de Weerdt).
[1] This article is based on the Master's thesis work of Baart (2018). The authors contributed equally to this work.

https://doi.org/10.1016/j.ejor.2020.09.042

may have sequence-dependent setup times (which in our model start after the release time) in addition to their processing time p_j, and $\bar{d}_j$ means that the completion time C_j of j should meet the following condition for j given a preceding completion of job i: $\max\{r_j, C_i\} + s_{ij} + p_j \leq C_j \leq \bar{d}_j$. We use 0 to denote the index of a dummy first job such that s_{0j} is the setup time for job j in case it is the first job. The term "reject" is less common, but is used for example by Zhang, Lu, and Yuan (2009) to denote that any job j can be rejected at a "penalty v_j"; the notation R indicates the set of rejected jobs. The other element in the (minimization) objective function in the three-field notation is the total weighted tardiness of all non-rejected jobs, i.e., $\sum_{j \notin R} w_j T_j$ where $T_j = \max\{C_j - d_j, 0\}$ with $d_j \leq \bar{d}_j$ being a due date.

In this paper we use the positive interpretation of order *acceptance* and scheduling (OAS), where we aim to maximize the revenue of accepted jobs $\sum_{j \notin R} v_j$ minus their tardiness penalty $\sum_{j \notin R} w_j T_j$. This problem is equivalent to the minimization objective except when analyzing the approximation ratio. This can be seen for example when almost all jobs can be scheduled on time: the value of the optimal solution in our formulation is then close to $\sum_j v_j$ and the performance ratio is the approximate value divided by the value of the optimal solution (so a value just below 1), while for the minimization objective the value of the optimal solution is close to 0, and the performance ratio for an instance would be computed by the approximated result divided by the optimal value, which may give quite a large ratio (possibly infinite).

The results in this paper immediately also hold when the tardiness objective and/or the sequence-dependent setup times are left out. Without total tardiness, the problem can be described by $1 \mid r_j; s_{jk}; \text{reject}; \bar{d}_j \mid \sum_{j \in R} v_j$ and is referred to as the Orienteering Problem with Time Windows (OPTW) (Gunawan et al., 2016; Labadie et al., 2012). Additionally removing the sequence-dependent setup times occurs in the literature as the Job Interval Selection Problem (JISP) or the Throughput Maximisation Problem (Kolen, Lenstra, Papadimitriou, & Spieksma, 2007), which is described by $1 \mid r_j; \text{reject}; \bar{d}_j \mid \sum_{j \in R} v_j$.

As OPTW and JISP are special cases, OAS is clearly NP-hard. A minimal hard special case with rejection is $1 \mid \text{reject}; \bar{d}_j \mid \sum_{j \in R} v_j$, to which the knapsack problem can be straightforwardly reduced. Finding a maximal polynomially solvable case is non-trivial, as rejection is not considered in the complexity hierarchy for scheduling problems by Brucker (2007). There is, though, a strong relation between the objective of weighted tardiness and of rejection with revenue: define a "lateness" penalty function that is 0 when the job finishes before the due date, linearly increases with lateness until the job revenue is reached and then stays constant at the job revenue. As a further restriction to $p_j = p$ enables polynomial algorithms for total tardiness (i.e., $1 \mid p_j = p \mid \sum_j T_j$ by Baptiste (2000)), this is a good candidate for a polynomial special case of OAS. Providing the respective polynomial algorithm, however, would also close a long-standing open problem for (regular) weighted total tardiness (Brucker, 2007), which is not the focus of the current contribution.

1.2. Example

An example OAS problem instance consisting of 4 jobs is shown in Table 1. In this problem there is no time jobs 1 and 4 both could start, so there are at most 3 jobs with overlapping (start) time windows. Since furthermore there is a time at which exactly 3 jobs can start (e.g., at 3), we say the width of this problem is 3. The optimal sequence is $1 \to 3 \to 4$, with completion times at 4, 8 and 11, a rejection penalty of 3, and a total tardiness of 0. However, if this problem is seen as the first 4 jobs of a larger problem, then there are multiple sequences that may later lead to an optimal solution, as some with higher costs complete earlier, potentially allowing a better selection of subsequent jobs (with a high penalty). Fig. 1 illustrates the time windows (and thus the slack) of the jobs in the example.

Table 1
Example OAS problem instance consisting of 4 jobs. The setup times are not shown in the table; they are $s_{ij} = 1$ for all i, j except $s_{04} = s_{41} = s_{43} = 2$.

i	r_i	p_i	d_i	$\bar{d}_i$	v_i
1	0	3	7	8	2
2	2	2	9	10	3
3	3	3	8	10	4
4	6	2	13	14	2

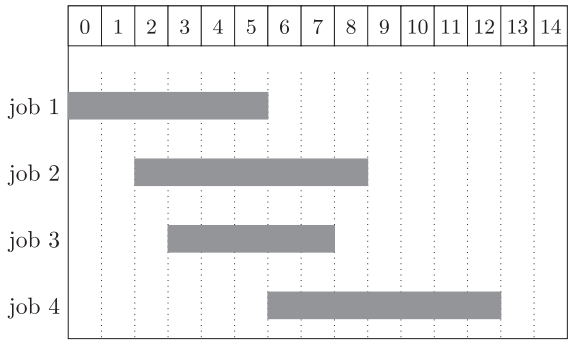

Fig. 1. The grey bars denote the availability windows of jobs in the example. The length of the grey bars represent the slack.

1.3. Related work

The aim of our research is to find algorithms that give guaranteed (near)-exact solutions without an exponential growth of the runtime in the size of the input, i.e., the number of jobs n. Such algorithms thus solve an instance of size n in $O(f(k_1, k_2, \ldots) \cdot \text{poly}(n))$ time for some computable, typically super-polynomial function f (Downey & Fellows, 2013). In this bound the exponential element is captured by this function f of one or more parameters $k_1, k_2, \ldots$ of the problem instances. Algorithms with such a property are called fixed-parameter tractable (FPT). If the degree of this poly(n) function is small, this is a stronger property than polynomial-time solvability for *constant* k: an algorithm running in time $O(n^k)$ can be impractical for large n even for small values of k.

A variant of the JISP without release times and with weighted completion time as objective (instead of tardiness) is FPT by taking any two out of the following three as parameters: the number of distinct processing times, the number of distinct penalties, and/or the maximum number of jobs to be rejected (Mnich & Wiese, 2015). If only the number of rejected jobs is the fixed parameter, the problem becomes W[1]-hard, which prohibits the existence of such a fixed-parameter algorithm unless FPT = W[1].

There are FPT results for scheduling problems without allowing rejection. These use other parameters, such as the partial order width of the precedence relations (Fellows & McCartin, 2003), the number of machines, the looseness, and the slack for a multi-machine scheduling problem (van Bevern, Niedermeier, & Suchý, 2017), or the number of different due dates, of different processing times, of different weights, and of time windows overlapping in any point in time (w) (van Bevern et al., 2016). However, before the work presented in this paper, it was unknown whether OAS is FPT.

Besides considering FPT, approximation algorithms may provide desired guarantees. However, unless P=NP, there is no polynomial-time algorithm that guarantees a constant-factor approximation for OAS (Nobibon & Leus, 2011). A weaker negative result holds

for JISP: a polynomial-time 2-approximation exists, but there is no polynomial-time approximation scheme (PTAS), i.e., a family of algorithms with arbitrary precision where the approximation is at least $1-\epsilon$ times the optimal value (Spieksma, 1999) (for maximization). This also forbids the existence of a so-called fully polynomial-time approximation scheme (FPTAS), where additionally the runtime of the algorithm is polynomial in $\frac{1}{\epsilon}$ as well as in the input size.

1.4. Contributions

In spite of these negative approximability results from the literature, we provide an FPTAS for the OAS problem, which generalizes the problems discussed above: first, OAS is shown to be FPT in the parameters of the width w (i.e., the maximum number of overlapping time windows) and the slack σ, which is defined as $\max_j\{d_j - p_j - r_j\}$. In particular, we provide a fixed-parameter algorithm with a runtime bound of $O(n^2 \cdot w^2\sigma 2^w)$. The main two insights leading to this result are that a dynamic programming formulation can be given in which the state space is not exponential in n (but in w only), and that the state space can be further reduced w.l.o.g. by implementing a so-called dominance rule. From the viewpoint of recent work on using decision diagrams for optimization (Bergman, Cire, Van Hoeve, & Hooker, 2016), this dominance rule can be seen as a special case of a state merging that is without loss of information, and can be applied across different layers.

Second, using only w as a parameter, an FPTAS is presented with a runtime bound of $O(\frac{n^3 \cdot w^2 2^w}{\epsilon^2})$. To the best of our knowledge, this is a first fixed-parameter FPTAS for scheduling problems: for every $\epsilon > 0$, the problem can be $(1-\epsilon)$-approximated in time $O(f(w) \cdot \text{poly}(n, \frac{1}{\epsilon}))$.

This parameter w limits the number of jobs that are available at the same time. This width is smaller than the total number of jobs in many real-life situations where the length of time-windows is smaller than the problem horizon, for example in satellite scheduling (where jobs are only available when the satellite is near the location in its orbit (He, de Weerdt, & Yorke-Smith, 2019b)), or vehicle routing (where sometimes time slots for delivery are given in minutes while the schedule is in hours (Qureshi, Taniguchi, & Yamada, 2009)). For more general instances, we provide a neighbourhood heuristic enforcing such a limited width artificially.

All three algorithms are evaluated against a recent successful exact branch-and-price method (Silva, Subramanian, & Pessoa, 2018) as well as state-of-the-art heuristics HSSGA (Chaurasia & Singh, 2017), ILS (Silva et al., 2018) and Tabu-Based Large Neighbourhood Search (He, de Weerdt, & Yorke-Smith, 2019a). The exact method finds optimal solutions for instances that had not been solved to optimality before, which also contributes to our understanding of the quality of heuristics.

2. Background

Apart from the literature mentioned in the introduction, there are a number of related results on exact and approximate algorithms. We also discuss state-of-the-art heuristics in this section.

2.1. Exact approaches to similar problems

An approach that is close to our exact algorithm is that of finding the maximum (weighted) independent set in a so-called strip graph (Halldórsson & Karlsson, 2006). Such a strip graph can encode a JISP: each possible start time of a job is represented by a vertex and the vertices of each job form a clique. Vertices from jobs with overlapping time intervals $[t_j, t_j + p_j)$ are also connected. Each job-start-time vertex can be assigned a weight v_j. The maximum (weighted) independent set in this graph then represents the feasible schedule with the most (valuable) jobs. Finding this with dynamic programming can be done in runtime linear in the number of vertices in this graph and exponential in the width. This specific graph encoding does not encode sequence-dependent setup times and minimizing the total weighted tardiness.

Other exact approaches are based on mixed-integer program formulations. For example Silva et al. (2018) have shown that a branch & price method outperforms a time-indexed formulation (TIF). Li and Ventura (2020) have run benchmarks of size up to 30 jobs to show that a dynamic programming-based solution for OAS, but without release times also outperforms TIF.

Additionally removing the possibility of rejection from the problem, and minimizing the (weighted) number of tardy jobs, Hermelin, Karhi, Pinedo, and Shabtay (2018) show that this problem variant is FPT for any two out of three parameters. These parameters are the number of different due dates, processing times, and weights in the set of input jobs.

2.2. Decision diagrams

A recent efficient approach to solving sequencing problems uses a representation called *decision diagrams* (Cire & van Hoeve, 2013; Hooker, 2017). This representation has strong similarities to (bounded) search trees as well as to dynamic programming (Woeginger, 2003): as in search trees, the state space is represented by a graph where vertices represent states and edges represent decisions; similar to dynamic programming, whenever decisions lead to states for which further decisions are equivalent, these can be represented and considered only once.[2] This leads to a directed acyclic graph with two special vertices: a vertex without incoming edges, the root s, representing the complete problem, and a vertex without outgoing edges, the sink t, representing the state where no further decisions have to be made. The paths in this graph from the root to the sink represent (all) feasible solutions. When weights on the edges represent costs, the shortest $s-t$ path represents an optimal solution. See the book by Bergman et al. (2016) for a more complete introduction.

Also for single-machine scheduling (but without allowing rejection), this approach has been quite successful: from root to sink, each edge represents the next job to be scheduled, resulting in a diagram with as many so-called *layers* as the number of jobs n, and at most $O(2^n)$ states in one layer (i.e. the *width*) in the exact representation. Cire and van Hoeve (2013) define a way to partition states in a layer and merge them accordingly in order to restrict the size of the diagram to a certain width. They have integrated this approach into a constraint programming solver, and have shown that this can be used to solve problems where jobs have precedence constraints as well. Hooker (2017) defines state merging heuristics for nodes in the same layer based on finish time and shortest path and compares these experimentally.

From the viewpoint of decision diagrams, our paper shows (1) how to extend this approach to include rejection of jobs, (2) that it is then useful to merge nodes from different layers, (3) that the size/width of the diagram then is exponential only in parameters slack and the maximum number w of overlapping time windows, and polynomial otherwise, (4) that this thus implies existence of a fixed-parameter tractable algorithm, (5) that some states dominate others and the dominated states can be removed without loss of optimality, and 6) there exists a node merging operator that constructs a restricted diagram of size exponential only in w that can

[2] Although in general finding out which states are equivalent could be NP-hard (Kinable, Cire, & van Hoeve, 2017).

Table 2
Runtime (in seconds, showing the four most significant digits) of exact methods on instances by Cesaret et al. (2012) with 50 and 100 jobs and $\tau = 0.9$. The best average runtime is highlighted in bold. A '-' indicates that the runtime limit of 3600 seconds is met. If not all 10 instances were within the time limit, the number of successful instances used to compute the average is given in parentheses.

				Exact method (EM)			EM without domination			B&P
n	R	σ	w	Min	Avg	Max	Min	Avg	Max	Avg
50	0.1	89	10.1	0.45	**0.7245**	1.22	2.239	3.838	6.459	304.6
	0.3	141.6	11	0.448	**1.568**	2.36	3.047	11.76	17.76	15.70
	0.5	190.8	12.8	1.991	**9.282**	47.32	16.97	61.92	197.6	76.10
	0.7	251.9	14.5	3.571	**27.61**	112.1	38.08	272.2	1334	54.00
	0.9	325.6	15.5	4.996	280.9	1522	30.59	653.8 (8)	-	**238.0**
100	0.1	166.4	18.9	123.6	**313.0**	833	868.1	1966 (9)	-	-
	0.3	275.6	18.7	158.7	**919.8**	2783	1336	2100 (3)	-	-
	0.5	392.5	20.6	694.5	**1373** (5)	-	-	-	-	-

guarantee an arbitrary bound ϵ on the optimality of the solution in runtime polynomial (except for w) in the input and ϵ, i.e., a fully polynomial-time approximation scheme in parameter w.

2.3. Approximation algorithms for similar problems

For OAS without sequence-dependent setup times, deadlines, and with a makespan objective instead of tardiness, Zhang et al. (2009) provide a dynamic program with runtime $O(n\sum_j v_j)$. They use the result that $1|r_j|C_{max}$ can be solved by considering jobs ordered on earliest release date (Lawler, 1973). This leads to a 2-approximation in $O(n^2)$ and a FPTAS in $O(\frac{n^3}{\epsilon})$.

For JISP (or, as they call it, the 1-machine throughput maximisation problem), Berman and DasGupta (2000) provide an FPTAS providing a $\frac{2}{1-\epsilon}$-approximation in $O(\frac{n^2}{\epsilon})$, along results for multiple machines.

2.4. Heuristics

Exact approaches to the OAS (and variants) are notorious for being feasible only for small problem sizes. For example, for OAS where a subset of jobs is required to be scheduled, Nobibon and Leus (2011) show that a state-of-the-art time-indexed formulation is unable to solve one-third of a set of instances of size $n = 40$ to optimality in under two hours. Their advanced two-phase branch-and-bound approach reaches this limit for 1 out of 18 instances of size $n = 50$. More recently, Silva et al. (2018) show that for $n = 100$ almost no instance is solved to optimality within a time limit of one hour (for four different approaches).

In situations where larger problems need to be solved, or decisions need to be taken within seconds rather than hours, inexact approaches are used. We discuss here the three most recent successful heuristics for OAS. First, in the hybrid steady-state genetic algorithm (HSSGA) by Chaurasia and Singh (2017) in each generation only the worst member of the population is replaced by a single offspring. This offspring is produced using multi-point crossover that maintains a subset of jobs and their relative ordering, and subsequently inserts other jobs greedily, also considering the setup times. Additionally, a local search procedure swapping jobs further aims to improve this candidate solution. HSSGA produces solutions to a set of benchmark problem instances of $n = 100$ that are between 1 and 12% of an upper bound, in on average about 12 seconds. These results are significantly better than Tabu Search (Cesaret, Oğuz, & Salman, 2012) and the Artificial Bee Colony algorithm (Lin & Ying, 2017), and marginally better than an evolutionary algorithm proposed in the same paper (EA/G-LS).

Second, based on a series of papers using iterated local search for scheduling with sequence-dependent setup times by Subramanian and Farias (2017), a multi-start algorithm is proposed specifically for OAS, which is denoted by ILS (Silva et al., 2018). ILS also significantly outperforms Tabu Search (Cesaret et al., 2012), producing better solutions in 60% of the instances compared to failing to do so in only 7.6%. The best solutions found by ILS are also better than those found by DRGA, GA, HH, and LOS, as reported by Silva et al. (2018), using results from Nguyen (2016).

Finally, He et al. (2019a) report that a hybrid method of Tabu Search and Adaptive Large Neighborhood Search, denoted by ALNS/TPF, outperforms ILS as well as a simpler hybrid by Liu, Laporte, Chen, and He (2017) on the same set of instances.

3. An exact dynamic programming method for OAS

In this section an exact method for OAS is proposed. Throughout we assume without loss of generality that the deadline $\bar{d}_j$ for each job j is set such that including the job right before this deadline, so with completion time $\bar{d}_j$ and thus tardiness $T_j = \bar{d}_j - d_j$, does not have higher total costs than excluding it, i.e., $w_j \cdot T_j \leq v_j$. The exact algorithm is based on the following recursive formulation of the maximum value of a solution OPT(i, X, t_i) for the subproblem that remains after i has been scheduled (as the last job) at start time t_i, and where X contains all jobs j that could have been scheduled at (or later than) $C_i + s_{ij}$, but have already been scheduled. These jobs X are excluded from the set of all jobs $F(i, t_i)$ that can be feasibly scheduled immediately after i. The value of scheduling a job $k \in F(i, t_i)$ next is the sum of its revenue v_k minus the (possibly 0) contribution to the weighted tardiness, and the effect of this decision on the remainder of the schedule:

$$\mathrm{OPT}(i, X, t_i) = \max_{k \in F(i,t_i) \setminus X} \{v_k - w_k \cdot T_k + OPT(k, X_k, t_k)\} \quad (1)$$

where

$$\begin{aligned} t_k &= \max\{C_i, r_k\} + s_{ik} \\ X' &= X \cup \{k\} \\ X_k &= X' \setminus \left\{ j \in X' | | \bar{d}_j - p_j < C_k + s_{kj} \right\} \\ T_k &= \max\{C_k - d_k, 0\} \\ F(i, t_i) &= \left\{ j | | C_i + s_{ij} \leq \bar{d}_j - p_j \right\} \end{aligned} \quad (2)$$

When $F(i, t_i) \setminus X = \emptyset$ then $\mathrm{OPT}(i, X, t_i) = 0$. As before, we use C_k to denote the completion time of job k, i.e., $t_k + p_k$. The optimal costs are then given by OPT(0, ∅, 0).

The most important idea of this dynamic program is encoded in the use of the set X, which we call the (to be) *excluded jobs*. In a given state (i, X, t_i), the set X represents jobs that have already been scheduled and can thus not be selected next. To prevent an exponential state space with all such possible subsets of jobs, this set is kept smaller by restricting it to jobs for which the time windows allow scheduling them in the remaining subproblem. This is formally expressed in the definition of X_k where jobs are removed for which the latest possible start time is before the time at which the first job in the remaining subproblem can start. This significantly reduces the runtime bound.

The formulation in Eq. (1) only considers feasible sequences of jobs, because at every recursive step: (i) only jobs are considered (in $F(\cdot,\cdot)$) that can be completed before their deadline, (ii) the selected job k is scheduled at a time t_k that is guaranteed to be after its release time as well as after the completion time of the previous job, and the sequence-dependent setup time is considered, and (iii) no job is considered twice, because any selected job is included in X until its latest start time. Furthermore, the value resulting from this equation is optimal, because i) all possible next feasible jobs are considered, and ii) the schedule time t_k of the next job k is the earliest time it can be scheduled (after job i) and this is never worse than waiting, both because of the tardiness costs as well as allowing for more jobs when completing earlier.

Theorem 1. *A dynamic programming implementation of the recursive function specified in Eqs. (1) and (2) is a fixed-parameter tractable algorithm with parameters w and σ, and has a runtime bound of $O(n^2 \cdot w^2\sigma 2^w)$.*[3]

Proof. Let w_t be the number of jobs j that include t in their availability interval, i.e., with $t \in [r_j, \bar{d}_j - p_j]$ and let $w = \max_t w_t$, then $|X_k| \leq w$ for all k by definition. Furthermore, let σ_j be the *slack* of job j, i.e., the number of allowed different starting times for integer domains: $\sigma_j = \bar{d}_j - p_j - r_j$ and let $\sigma = \max_j \sigma_j$. The state space then is $O(n \cdot \sigma 2^w)$, as (i) there are n possible jobs i, (ii) for each job σ possible starting times t_i, [4] and (iii) for each of these (i, t_i) pairs, there are at most w jobs that could be scheduled after i, which gives a total of 2^w possible subsets of jobs X.

To arrive at an efficient runtime, in the implementation we preprocessed the time windows of jobs to find periods in which they are relevant and used a priority queue to go over all states in order of the start time of the last job. Further, we see that k can take at most $O(n)$ possible values in each step and it takes $O(w)$ work to compute the respective set of jobs. Further it takes at most log of the number of states to update the priority queue, which is $O(\log n + \log\sigma + w)$, thus $O(w)$ in all cases except when w is very small (less than $o(logn)$ or $o(log\sigma)$). The runtime thus is $O(n^2 \cdot w^2\sigma 2^w)$, which is indeed FPT with w and σ as parameters. □

Moreover, this dynamic programming algorithm is FPT with just w as a parameter if the slack is less than the number of jobs; if release times and deadlines are given in unary it is pseudo-polynomial FPT.

4. Multi-valued decision diagrams and state dominance

The state space for the FPT algorithm can be reduced by taking the perspective of a decision diagram: we represent each state, i.e., each combination of arguments (i, X, t_i) that occurs as a consequence of Eq. (1) by a vertex, and each $k \in F(i, t_i)\backslash X$ as an edge with (nonnegative) weight $v_k - w_k \cdot T_k$. We define the value of a state $v(i, X, t_i)$ to be the length of the longest path P from the root node to (i, X, t_i). In this diagram we can indicate some states that are not essential for finding the optimal solution.

Definition 1. State $s_1 = (i_1, X_1, t_1)$ *dominates* state $s_2 = (i_2, X_2, t_2)$ if $i_1 = i_2$, $t_1 \leq t_2$, $v(s_1) \geq v(s_2)$, and $X_1 \subseteq X_2 \cup \left\{j \in X_1 \middle| \bar{d}_j - p_j \geq C_{i_2} + s_{i_2 j}\right\}$.

Proposition 1. *If a state s_2 is dominated by another state s_1, then if there is an optimal solution having s_2 as a sub-problem, there is also an optimal solution containing s_1.*

The proof of Proposition 1 is straightforward as state s_1 puts fewer restrictions than s_2 on subsequent sequences, so any solution starting from s_2 can also be started from s_1. Since s_1 has at least as high a value as s_2, the latter is not required for an optimal solution.

For any fixed job j and (to be) excluded jobs X, we thus can ignore states (j, X, t) with both a later start time and lower accumulated value than any other state. This is a special case of a state merging operator where all dominated states are merged with the dominating state, and where the involved states are not necessarily part of the same layer.

In this paper we refer to the exact algorithm that uses this dominance rule as the Exact Method (EM).

5. Approximate solutions for OAS

The runtime of the exact method is bounded by a function that has the slack σ as a factor and depends exponentially on the width w. In this section we show how the exact formulation can be used in two different approximation methods: the factor *sigma* can be removed to obtain a FPTAS, and the width can be bounded, leading to a heuristic with an efficient runtime but without guarantee on the value.

5.1. An FPTAS for OAS

The effect of the slack on the runtime is removed by an approximate state merging operator that merges states with finish times t and values that are close enough (for fixed j and X).

Definition 2. For a given job j and excluded jobs X, the *time-value Pareto front* $P(j, X)$ is the set of states (j, X, t) that are not dominated (as defined in Definition 1).

The idea of the ϵ-approximate state merging is to partition each set $P(j, X)$ into at most $\frac{n}{\epsilon}$ subsets with similar time and value, and remove all states within each subset except for the one with the lowest value (and thus smallest possible time).

Definition 3. For a given job j and excluded jobs X, the *ϵ-approximate Pareto front* $P_\epsilon(j, X)$ is a set of states defined as follows. Let $v_{\min} = \min_{s \in P(j,X)} v(s)$, $v_{\max} = \max_{s \in P(j,X)} v(s)$, $\Delta = v_{\max} - v_{\min}$, and $\delta = \frac{\epsilon\Delta}{n}$. Then sort states in $P(j, X)$ on their value, and make $\left\lceil \frac{\Delta}{\delta} \right\rceil = \lceil \frac{n}{\epsilon} \rceil$ sets of states such that states within a set differ at most δ in value. For each subset of the partition, 'merge' all states into the state with the smallest possible time t (and thus the lowest value), and remove all others.

Theorem 2. *The maximum value solution when considering only states in the ϵ-approximate Pareto front is at least $1 - \epsilon$ times the optimal solution, and the runtime is bounded by $O(\frac{n^3 \cdot w^2 2^w}{\epsilon^2})$.*

Proof. Each solution, represented by a path through the state space (or decision diagram), intersects with at most n Pareto fronts $P(j, X)$. Because of the ϵ-approximation of such a front, this intersection involves a state with an earlier completion time, which has at least as many subsequent paths as any other vertex with the same j and X. Further, this state has at most $\frac{\epsilon(v_{\max} - v_{\min})}{n}$ less value than the state used in the optimal solution. Given that a state with a value $v_{\max}$ can be extended trivially to a full solution (by not adding any more jobs), the value of the optimal solution OPT^* is always at least $v_{\max}$. Consequently, the total loss for all n such intersections on a path together is bounded above by $\epsilon(OPT^* - v_{\min})$, which is less than ϵOPT^*, because $v_{\min}$ is non-negative. Therefore the outcome of the approximation is at least $(1 - \epsilon) \cdot OPT^*$.

Regarding the runtime, due to the ϵ-approximation, the number of states with the same job and excluded-jobs state is now limited to $O(\frac{n}{\epsilon})$, whereas this is $O(\sigma)$ in the exact algorithm. This gives a

[3] Unless w is $o(logn)$ or $o(log\sigma)$.

[4] Although this may be significantly less, since (1) σ is an upper bound of possible starting times over all jobs, or (2) when r_i is larger than completion times of preceding jobs – since then the optimal starting time would be r_i.

bound on the number of states of $O(\frac{n^2 \cdot 2^w}{\epsilon})$. Calculating the next states remains at a cost of nw as before, but maintaining the ϵ-approximation involves an extra $O(\frac{n}{\epsilon})$ insertion procedure for each new vertex. The resulting algorithm therefore has a runtime bound of $O(\frac{n^3 \cdot w^2 2^w}{\epsilon^2})$. □

Consequently, the proposed algorithm including the ϵ-approximation of the Pareto front is an FPTAS.

5.2. Neighborhood heuristic

The ideas presented in the previous section can also be used for problems with a larger width w by introducing an artificial neighborhood of size $\hat{w} \leq w$, similar to the Balas-Simonetti neighbourhood used for traveling salesman problems (Balas, 1999; Balas & Simonetti, 2001; Gutin & Punnen, 2006), but now adapted to OAS, using the EM. The idea is to define an artificial order of the jobs and only allow solutions where subsequent jobs are close to each other in this order.

For OAS we define an approximate algorithm, which we call "Balas", as follows: we order on latest start time $\bar{d}_j - p_j$, and ensure that any two subsequently scheduled jobs i and j are within distance $\hat{w}$ in this order, i.e., $|i - j| \leq \hat{w}$. We consequently modify the dynamic program presented earlier as follows: (1) instead of considering all jobs $k \in F(i, t_i) \backslash X$ that can be started, we consider only the jobs in the respective set that come at most $\hat{w}$ before i when ordered by latest start time or at most $\hat{w}$ after i. (2) The set X_k from Eq. (2) can be limited to jobs with indices differing at most $\hat{w}$ from k. The runtime of this Balas heuristic is therefore $O(n\hat{w}^3 \cdot \sigma 4^{\hat{w}})$. This algorithm provides a solution without any approximation guarantee. Its performance is evaluated in the next section.

6. Experimental evaluation

The contributions of this paper are mainly theoretical, contributing to understanding the structure of the OAS problem, and in particular the role of the width and the slack. However, for solving such problems in practice, it is very relevant to know whether these ideas by themselves are sufficient to outperform the state-of-the-art, developed over the past decade (since Oguz et al., 2010), and if so, for which problem instances – with which properties (among which, of course, the width, given its exponential influence on the bound on the runtime). This additional knowledge supports the selection of an algorithm for specific use cases.

In this section we therefore present experimental evidence answering the following questions regarding the algorithmic ideas presented in this paper:

1. Is the effect of the dominance rule on runtime significant enough?
2. Does the exact method (EM) outperform the state-of-the-art in exact methods, and if so, for which problem instance properties?
3. Do the approximate solutions (FPTAS and Balas) outperform state-of-the-art heuristics, and if so, for which problem instance properties?
4. How does the performance of the new algorithms depend on the *width* of the problem instances, also compared to other algorithms?
5. How does the performance of the new algorithms depend on the *correlation* of revenue and processing time, also compared to other algorithms?

Regarding these problem properties we expect (1) that instances where some jobs with a short processing time have a high revenue, and some with a long processing time have a low revenue are easier for the algorithms than when revenue and processing time are correlated, and (2) that the width significantly influences the runtime. To verify these hypotheses, we include two more specific experiments to answer the following questions.

Questions 1–3 are answered using the standard benchmark set for OAS from Cesaret et al. (2012); for questions 4 and 5 we have generated two new benchmark instances: one where the width is varied across the instances, and one where the correlation is varied.

6.1. Benchmark problem instances

First, the standard OAS benchmark, used in several papers after its first appearance (2012) contains instances with sizes of 10 to 100 jobs, randomly generated using a fixed procedure using two parameters: a tardiness factor τ and a due date range R (based on an earlier model for scheduling problems (Beasley, 1990; 2018)). Processing times p_j are drawn uniformly from a fixed range (of [0,20]), release times r_j from the interval $[0, \tau \Sigma_j p_j]$, sequence-dependent setup times s_{ij} from [1,10], and job revenues v_j from [1,20]. For each due date d_j, the interval from which it is drawn is related to R, and defined by $[1 - \tau - \frac{1}{2}R, 1 - \tau + \frac{1}{2}R] \cdot (\sum_j p_j + \max_i s_{ij} + r_j)$. Any job with an infeasible combination of release time and due date is removed. This leads to problem instances such as illustrated in Fig. 2.

Analyzing the width of the instances in this artificially generated benchmark set, we see that for $\tau \leq 0.7$, each instance has some time points which are included in the feasible time window of at least half of the jobs, i.e., $w \geq \frac{1}{2}n$. The methods proposed in this paper have a runtime (and memory use) exponential in w, and indeed do not even complete in less than an hour for these instances. Here we therefore present results only for a subset with a width up to about 25–30% of all jobs: for $\tau = 0.9$ the instances have a width of 10–16 for $n = 50$, and of 18–26 for $n = 100$.

Second, the width-based benchmark set was generated for $R = 0.1$ and width w from 3 to 19. Here we chose a wider range of time values, scaling by $\alpha = 2^{10} = 1024$. To explore only the effect of width, the slack σ is drawn uniformly from a fixed range of [2406, 2714]. Processing times p_j are drawn uniformly from a fixed range of $[1, 2\alpha]$, and sequence-dependent setup times s_{ij} from $[1, 10\alpha/n]$. Job revenues v_j are set equal to p_j to create more challenging instances (i.e., fewer obviously dominated jobs). Release times r_j are set as early as possible, considering the jobs in the order they are generated, without exceeding the constant width. Deadlines then follow because $\bar{d}_j = \sigma_j - (r_j + p_j)$, and due dates d_j are set to $\bar{d}_j - Rp_i$. Further, penalty weights are such that the value becomes zero when $C_j = \bar{d}_j$. A consequence of fixing the slack is that instances with a large width have jobs with long processing times and thus can include fewer jobs in the optimal solution. For each width, 5 such instances are created, each containing $n = 100$ jobs.

Third, the correlation benchmark set was generated using a parameter (c), which defines the desired correlation between job revenue and processing time. The procedure is similar to the one for the width-based benchmark (for a width of 7). Here $\alpha = 2^{40}$, and the slack σ_j depends on R and is drawn from $\left[\frac{5}{2}\alpha(1 - R) + \alpha, \frac{5}{2}\alpha(1 + R) - \alpha\right]$. The setup times s_{ij} are drawn uniformly from the fixed range $[1, \alpha(1 - c)]$ and job revenue v_j from the real range $\left[cp_j + (1 - c)\alpha, cp_j + (1 - c)\alpha\right]$. The bench-

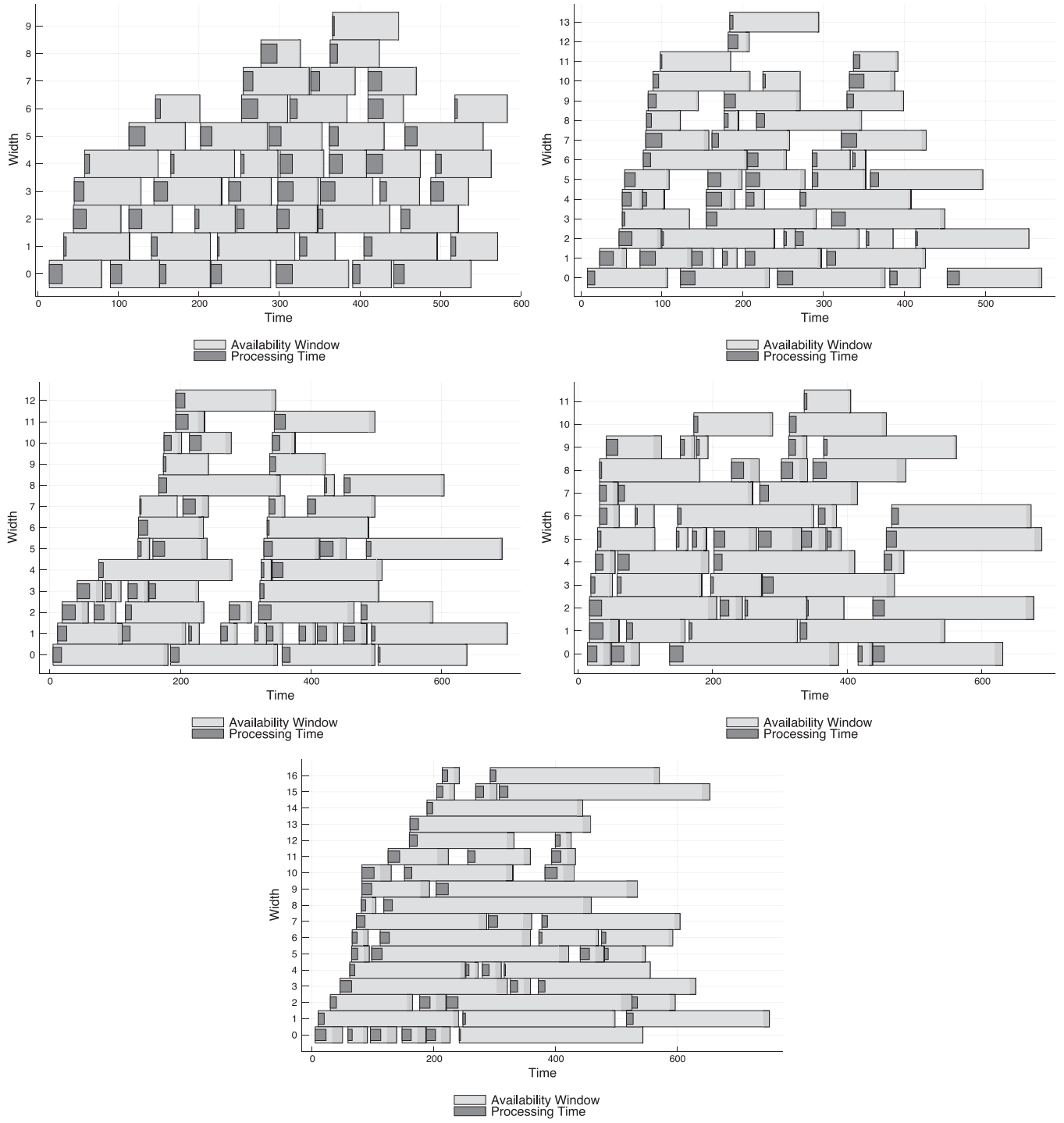

Fig. 2. Example problem instances of OAS for $n = 50$, $\tau = 0.9$, and $R = 1$, 3, 5, 7, and 9.

mark data as well as the implementation of the methods introduced in this article are publicly available.[5]

6.2. Exact methods and the effect of the dominance rule

The state-of-the-art exact method we compare to is the branch-and-price (B&P) approach by Silva et al. (2018). In their recent paper, this method is shown to outperform a number of other exact methods. Since their method is not publicly available we have included the runtime results they reported for the machine they used, an Intel i7-2600 with 3.40 gigahertz and 16 gigabyte of RAM with a time limit of 3600 seconds.[6] The machine used in our experiment is a comparable Intel Core i5-3470 3.20 gigahertz CPU with 8 gigabyte memory.[7] Runtime results (using a single thread) can be found in Table 2. For each combination of parameters (each row) 10 different instances were generated. The smallest (Min) and

[5] https://doi.org/10.5281/zenodo.4048462.

[6] The authors were so kind to inform us of the precise model over email.

[7] According to https://www.cpubenchmark.net/singleThread.html the CPU we used has a single-thread rating less than 1% lower than the i7-2600.

largest (Max) runtimes (in seconds) are reported, as well as the average over all 10 runs (Avg). When not all 10 runs are completed within the time limit, the number of completed runs is given within parentheses. For $n = 100$ with $R = 0.7$ and $R = 0.9$ both methods needed more than 1 hour of computation time, so those results are left out of this table.

From these results we observe that almost for all instances EM is faster than B&P, sometimes significantly so (e.g., for $n = 100$ optimal solutions were found for some instances in less than 3 minutes where B&P always hit the time limit of one hour). This answers the second question regarding EM outperforming the state-of-the-art positively. We also see that the dominance rule reduces the runtime significantly (answering the first question), so we enable it in all further experiments.

6.3. Heuristic methods

To answer the third question, regarding the performance of the approximation algorithm FPTAS and the Balas heuristic, we give runtime and approximation (gap to optimal) results for FPTAS, Balas, as well as for a number of state-of-the-art heuristic methods on these same benchmark instances. For the Balas heuristic we include results for two different values for $\hat{w}$: 5 and 12. To represent the state-of-the-art we include the most recently published heuristics: HSSGA by Chaurasia and Singh (2017), ILS by Silva et al. (2018), and ALNS/TPF by He et al. (2019a). Only for the latter we had access to the original source code. The implementation of the other two algorithms was done by one of our students, based on the description in the respective publications. Results for HSSGA were very close to those in the original paper. However, the results of ILS were not consistent with the originally reported results, so we indicate the results presented here by ILS*.

Because for these heuristics runtime and quality on the same instance may differ slightly from one run to the next, each presented result is the average of ten runs on the same instance, so the averages in the table are based on 100 runs. Table 3 presents these runtimes, showing that the presented FPTAS is competitive only for instances with a small width.

The quality of the results can be derived from Table 4. In the literature usually the gaps to an upper bound were reported, and these gaps were typically about 5 to 20% for these instances (Cesaret et al., 2012). Here, however, thanks to the exact method presented in this paper, we can now for the first time report the gap to the *optimal* solution. Surprisingly the gaps to optimality are rather small overall (around 1%), proving that state-of-the-art heuristic methods do really well and the upper bound used in the past is quite loose.

Regarding the FPTAS presented in this paper, although it has a theoretical guarantee on the quality, its gaps are larger than those from HSSGA and ALNS/TPF, and its runtime in many cases significantly so. Comparing runtimes to those of the exact methods, we see that, as can be expected, the heuristic methods are much faster (but of course the exact methods provide the optimal solution). However, there are actually a few instances where the exact method (EM) is faster than any of the state-of-the-art heuristic methods (for $n = 50$ with $R = 0.1$ or $R = 0.3$, see Table 2). In the case of the FPTAS this may seem even more surprising, since these methods are quite similar. However, the FPTAS has some overhead for identifying the ϵ-approximate Pareto-dominated states. Also the job revenues and processing times are not correlated in these instances, so many of the nodes that are merged in FPTAS are dominated anyway.

The performance of the Balas heuristic is much more interesting: Balas(5) (so with $\hat{w} = 5$) has a gap of about 5–10%, which is better than ILS*, but not as good as ALNS/TPF and HSSGA, which have a gap below 1%. However, it appears to be the fastest heuris-

Table 3
Runtime (in seconds) of approximate methods on instances by Cesaret et al. (2012) with 50 and 100 jobs, and $\tau = 0.9$. For slack σ and width w of these instances and runtimes of the exact methods, please refer to Table 2. The best average runtime is highlighted in bold.

		FPTAS ($\epsilon = 0.1$)			FPTAS ($\epsilon = 0.05$)			Balas (5)			Balas (12)			HSSGA			ILS*			ALNS/TPF		
n	R	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
50	0.1	0.618	0.956	1.59	0.71	1.09	1.91	0.2	**0.269**	0.421	0.319	0.476	0.909	3.74	3.89	4.01	2.8	3.18	3.76	2.77	2.96	3.07
	0.3	0.652	1.87	2.67	0.71	2.15	3.29	0.11	**0.302**	0.438	0.287	1.05	1.64	3.86	3.96	4.04	3.11	3.42	3.87	2.9	3.18	3.43
	0.5	2.36	9.04	32.8	2.71	11	45.5	0.227	**0.591**	2.37	1.13	5.91	36.9	3.75	3.97	4.18	3.88	4.47	5.1	3.25	3.55	3.93
	0.7	4.34	30	115	4.73	33	121	0.316	**0.814**	1.96	2.09	7.5	24.8	3.81	3.93	4.05	4.18	5.26	6.1	3.42	3.93	4.33
	0.9	5.25	269	1280	6.2	325	1660	0.3	**1.2**	3.73	2.39	29.3	103	3.87	4.04	4.35	5.48	6.13	7.03	3.93	4.5	5.43
100	0.1	154	358	960	179	453	1180	7.69	**10.1**	12.3	65.4	134	276	27.9	30.5	32.5	30.5	35.8	37.7	13.2	14	14.6
	0.3	196	1230	-	244	1130	3590	1.18	**4.72**	9.47	13.3	104	225	25.5	28.8	31.5	37.1	41.4	44.1	13.8	14.7	16.5
	0.5	886	1690	-	1040	2000	-	1.86	**5.39**	12.3	39.1	200	642	23.8	28.4	35.1	42.7	49.3	53.5	14.5	16.5	17.8

Table 4
Gap to the optimal solution (%) of approximate methods on instances by Cesaret et al. (2012) with 50 and 100 jobs, and $\tau = 0.9$. These optimal solutions were not known before, and are here provided by EM. The best average gap is highlighted in bold.

		FPTAS ($\epsilon = 0.1$)			FPTAS ($\epsilon = 0.05$)			Balas (5)			Balas (12)			HSSGA			ILS*			ALNS/TPF		
n	R	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
50	0.1	0.4158	2.397	4.208	0.404	1.12	1.95	0	0.847	2.22	0	**0**	0	0	0.0921	0.4	12.6	14.6	18.4	0	0.268	0.356
	0.3	0.9009	2.643	6.054	0.45	1.23	2.19	0.48	3	6.52	0	**0.109**	1.09	0	0.198	0.853	11.9	14.7	21.2	0	0.432	0.765
	0.5	0.9218	2.392	3.606	0.234	1.29	3.04	0.603	5.76	9.37	0	0.561	1.49	0	**0.132**	0.537	8.31	11.8	15.4	0	0.477	0.615
	0.7	1.803	2.877	4.303	0.549	1.22	1.85	2.05	6.3	10.7	0.0793	1.19	2.99	0.0205	0.22	0.577	8.98	12	13.9	0	**0.0886**	0.429
	0.9	0.3471	2.439	4.099	0.902	1.33	2.11	3.33	6.28	12.3	0	1.33	3.5	0	**0.191**	0.334	6.7	10.2	15.6	0	0.214	0.704
100	0.1	0.627	1.816	3.272	0.322	0.916	1.8	1.65	2.5	3.45	0	**0.153**	0.401	1.28	1.51	1.89	18.6	20.6	22.2	0.0501	0.198	0.627
	0.3	1.417	1.94	-	0.526	0.804	1.31	4.98	6.73	9.47	0.31	1.49	2.25	0.465	0.957	1.44	14.8	17.6	20.2	0.269	**0.174**	0.86
	0.5	0.8812	1.608	-	0.728	0.961	-	4.97	6.73	9.99	0.794	2.12	3.77	0.693	1.02	1.35	14.5	16.8	21.8	0	**0.293**	0.891

tic overall, with runtimes below 5 seconds even for 100 jobs. For $n = 50$ and $R = 0.1$ or 0.3, Balas(12) is both faster than the state of the art, and provides the smallest gap. For larger problems, the gap is comparable to the best performing heuristic, but its computation time increases up to 10 times that of the others.

6.4. Dependence on width

To study the effect of the width on the runtime (the fourth question), and find out more precisely under which conditions the new algorithms outperform the state of the art, we used the (for the methods in this paper) extremely challenging set of problem instances with a correlation of 1.0 and scaled values (with α), and increasing width $(3, 5, \ldots, 19)$ as described above. For the same set of parameters, 5 instances are generated randomly. A time limit of 1800 seconds is used. The results of heuristic methods are the average of 10 runs. The results are shown in Fig. 3.

For this benchmark set all methods except ILS and Balas(5) are within 1% of optimal. Considering the runtime, the new heuristic methods FPTAS and Balas(5) outperform all state of the art methods for instances with a width of 7 or less, which is very relevant for example in satellite scheduling: the maximum width for instances of size $n = 100$ in the AEOSS "Area" benchmark set is 7 and in the "Worldwide" set is 4 (He et al., 2019b). We further observe that Balas(5) is the fastest heuristics overall, even under extreme conditions (high width, high correlation, large α).

ALNS seems to be the all-round best performing heuristic, striking a good balance between high quality solutions and low computation time across all widths. If more time is available (2 minutes), then EM is the best choice. If on the other hand, a very quick response is required, Balas(5) is the best alternative.

6.5. Correlation between revenue and processing time

The final experiment (see Figure 4) confirms the hypothesis that a high correlation between job revenues and processing times in an instance increases the difficulty (runtime) for the algorithms presented in this paper, with the high values chosen here ($\alpha = 2^{40}$) in particular. We observe that the solution quality seems not to be significantly affected by the correlation except for ILS and Balas, which give better results for instances with a higher correlation. The runtime of all newly proposed methods are increasing with c, while all the others seem to be independent. This can be explained by the exact dominance rule being less effective with a high correlation. Still, runtimes for scheduling 100 jobs are well within an acceptable 6 seconds for all these instances and comparable to the state of the art, even for the highest correlation.

7. Conclusions and future work

The maximum number of jobs with overlapping time windows (the width w) is a parameter that has shown to be instrumental in designing an exact algorithm for single-machine scheduling with release times, deadlines and rejection (also known as order acceptance and scheduling, OAS). The resulting optimal algorithm has a runtime quadratic in the number of jobs n, linear in the slack σ (i.e., the maximum time any task can be delayed given its deadline), and exponential only in the width w. This shows that OAS is fixed-parameter tractable in w and σ. Requiring only w as a parameter, we can build on the same insights to arrive at an algorithm that approximates the optimum within a guaranteed factor of $1 - \epsilon$ and has a runtime bound of $O(\frac{n^3 \cdot w^2 2^w}{\epsilon^2})$, i.e., a fixed-parameter FPTAS. Based on the same principles, a neighbourhood heuristic can much more efficiently find very good solutions by fixing an artificial, but constant width. All three algorithms use a so-called dominance rule to reduce the state space when one partial

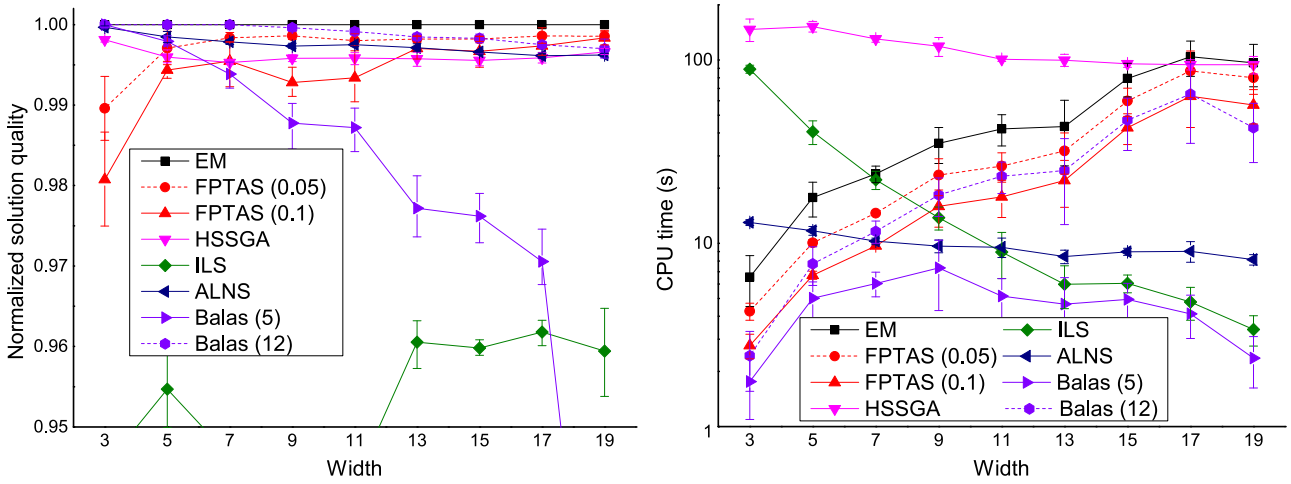

Fig. 3. Average solution value (left) and CPU time (right) versus width for the different algorithms on $n = 100$ with $\alpha = 1024$. The solution quality is normalized by the best value found for each instance. The error bars represent the 1st and 3rd quartiles.

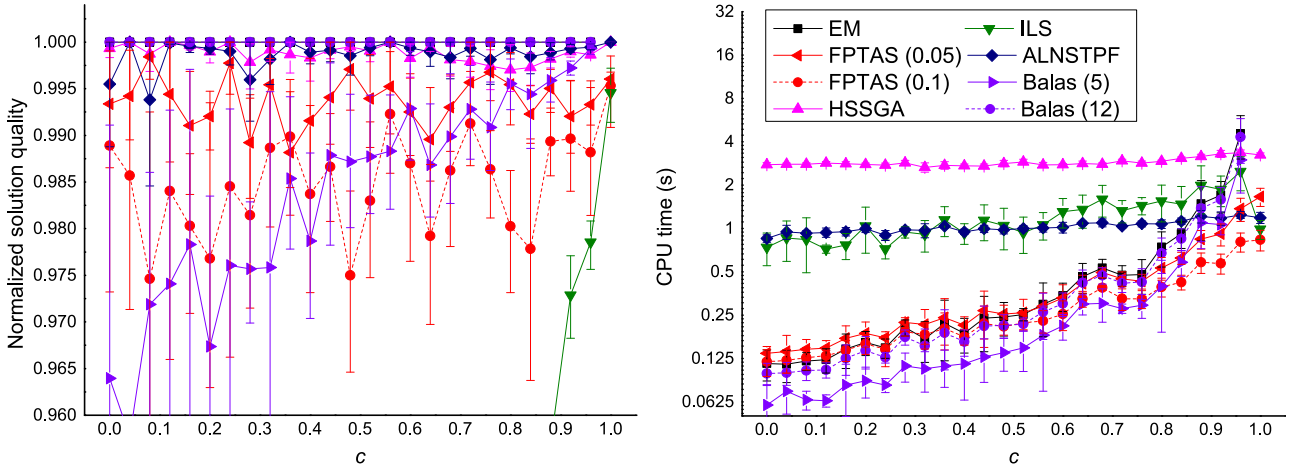

Fig. 4. Solution value (left) and CPU time (right) versus the correlation between the processing time and the revenue for the different algorithms on the covariance changing dataset with $n = 100$, $w = 7$, $\alpha = 2^{40}$. The error bars represent the 1st and 3rd quartiles.

solution cannot lead to a better solution than another partial solution. This is similar to the concept of state merging in recent literature on optimization using decision diagrams (Bergman et al., 2016), and can also be seen as a generalization of dynamic programming.

The three new algorithms have been experimentally evaluated on a standard benchmark set, as well as on custom-made instances to evaluate their performance depending on the width. For a number of benchmark instances of size $n = 100$ an optimal solution has been found for the first time. When the width is reasonably small (less than 15 for $n = 50$ and less than 21 for $n = 100$), the exact algorithm outperforms the state of the art. When benchmarked against recently published state-of-the-art heuristics, the approximation algorithm is only competitive for a smaller width (11 or less), but the Balas heuristic outperforms state-of-the-art heuristics under a wide range of conditions, depending on the choice for the parameter.

With these concrete results this paper additionally provides evidence that recent insights in decision diagrams and fixed-parameter analysis can be merged and generalized. First, the presented exact algorithm uses a multi-valued decision diagram (similar to for example Hooker, 2017), but does not impose the structure of layers: paths can have different lengths, and merges can occur between states at different distances from the root node. Second, the width of the decision diagram is bounded by a parameter of the input, thus making the resulting exact method fixed-parameter tractable. Third, the approximation algorithm defines a state merging operation that provides a guarantee on the performance. And finally, such ideas can be effectively used as a heuristic. An interesting avenue for further research is to find out for which other problems such a generalization of state merging in multi-valued decision diagrams and the analysis using parameters provides better insights and faster exact algorithms, or approximations with performance guarantees.

Conversely, recent results on decision diagrams and their use as relaxations and restrictions in a branch-and-bound search (Bergman et al., 2016) indicate a promising direction to tackle larger instances of OAS, building upon the model and dominance rule presented here.

Considering the approximation algorithm, admittedly, the experimental results indicate some directions for improvement. For example, the current design of the algorithm leads to an error that is close to ϵ, even if the total number of states is not that large. Merging could be done depending on the (estimated) state space instead. Second, once a solution has been found, the decision diagram could be used to improve upon it by considering whether using the merged ϵ-dominated states also leads to a feasible solution, but of higher quality. An alternative approach worthwhile

of investigating is to use the heuristic inside a local search approach (Hintsch & Irnich, 2018) such that it efficiently finds a locally optimal solution, and provide a bound for the overall optimisation problem. Further exploiting these insights may lead to an algorithm that can deal with even larger problem instances of OAS.

Finally, it is worth considering whether the presented fixed-parameter results can be extended to other problem classes, such as scheduling on parallel and uniform machines, as decision diagrams have been recently successfully applied to this more general problem as well (van den Bogaerdt & de Weerdt, 2018; 2019).

Acknowledgements

We thank Arthur Guijt for the implementation of HSSGA and ILS*, Pim van den Bogaerdt for proofreading our draft, Alin Dondera for running some extra experiments, and the CSC for their scholarship (201703170269) allowing Lei He to visit Delft.

References

Baart, R. (2018). *Algorithms for a bounded-width order acceptance and scheduling problem with sequence-dependent setup time using decision diagrams*. Delft University of Technology Master's thesis.

Balas, E. (1999). New classes of efficiently solvable generalized traveling salesman problems. *Annals of Operations Research, 86*, 529–558.

Balas, E., & Simonetti, N. (2001). Linear time dynamic-programming algorithms for new classes of restricted TSPs: A computational study. *INFORMS Journal on Computing, 13*, 56–75.

Baptiste, P. (2000). Scheduling equal-length jobs on identical parallel machines. *Discrete Applied Mathematics, 103*, 21–32.

Beasley, J. E. (1990). OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society, 41*, 1069–1072.

Beasley, J. E. (2018). Weighted tardiness (OR library). http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html, originally described in Beasley (1990) [Online accessed 26-Feb-2018].

Bergman, D., Cire, A. A., Van Hoeve, W.-J., & Hooker, J. (2016). *Decision diagrams for optimization*. Springer.

Berman, P., & DasGupta, B. (2000). Multi-phase algorithms for throughput maximization for real-time scheduling. *Journal of Combinatorial Optimization, 4*, 307–323.

van Bevern, R., Bredereck, R., Bulteau, L., Komusiewicz, C., Talmon, N., & Woeginger, G. J. (2016). Precedence-constrained scheduling problems parameterized by partial order width. In *Proceedings of the international conference on discrete optimization and operations research* (pp. 105–120).

van Bevern, R., Niedermeier, R., & Suchý, O. (2017). A parameterized complexity view on non-preemptively scheduling interval-constrained jobs - few machines, small looseness, and small slack. *Journal of Scheduling, 20*, 255–265.

Bianchessi, N., Cordeau, J.-F., Desrosiers, J., Laporte, G., & Raymond, V. (2007). A heuristic for the multi-satellite, multi-orbit and multi-user management of earth observation satellites. *European Journal of Operational Research, 177*, 750–762.

van den Bogaerdt, P., & de Weerdt, M. (2018). Multi-machine scheduling lower bounds using decision diagrams. *Operations Research Letters, 46*, 616–621.

van den Bogaerdt, P., & de Weerdt, M. (2019). Lower bounds for uniform machine scheduling using decision diagrams. In *Integration of constraint programming, artificial intelligence, and operations research* (pp. 565–580). Cham: Springer International Publishing.

Brucker, P. (2007). *Scheduling algorithms*. Springer Verlag.

Cesaret, B., Oğuz, C., & Salman, F. S. (2012). A tabu search algorithm for order acceptance and scheduling. *Computers Operations Research, 39*, 1197–1205.

Chaurasia, S. N., & Singh, A. (2017). Hybrid evolutionary approaches for the single machine order acceptance and scheduling problem. *Applied Soft Computing, 52*, 725–747.

Cire, A. A., & van Hoeve, W. J. (2013). Multivalued decision diagrams for sequencing problems. *Operations Research, 61*, 1411–1428.

Downey, R., & Fellows, M. (2013). *Fundamentals of parameterized complexity*. Springer.

Elffers, J., & de Weerdt, M. (2017). Scheduling with two non-unit task lengths is NP-complete. *Technical Report 1412.3095*. ArXiv.

Fellows, M. R., & McCartin, C. (2003). On the parametric complexity of schedules to minimize tardy tasks. *Theoretical Computer Science, 298*, 317–324.

Gunawan, A., Lau, H. C., & Vansteenwegen, P. (2016). Orienteering problem - a survey of recent variants, solution approaches and applications. *European Journal of Operational Research, 255*, 315–332.

Gutin, G., & Punnen, A. P. (2006). *The traveling salesman problem and its variations*. Springer Science Business Media.

Halldórsson, M. M., & Karlsson, R. K. (2006). Strip graphs - recognition and scheduling. In *Graph-theoretical concepts in computer science* (pp. 137–146). Berlin, Heidelberg: Springer Berlin Heidelberg.

He, L., de Weerdt, M., & Yorke-Smith, N. (2019b). Time/sequence-dependent scheduling: The design and evaluation of a general purpose tabu-based adaptive large neighbourhood search algorithm. *Journal of Intelligent Manufacturing*.

He, L., de Weerdt, M., & Yorke-Smith, N. (2019a). Tabu-based large neighbourhood search for time/sequence-dependent scheduling problems with time windows. In *Proceedings of the 29th international conference on automated planning and scheduling*.

Hermelin, D., Karhi, S., Pinedo, M., & Shabtay, D. (2018). New algorithms for minimizing the weighted number of tardy jobs on a single machine. *Annals of Operations Research*, 1–17.

Hintsch, T., & Irnich, S. (2018). Large multiple neighborhood search for the clustered vehicle-routing problem. *European Journal of Operational Research, 270*, 118–131.

Hooker, J. N. (2017). Job sequencing bounds from decision diagrams. In *Proceedings of the international conference on principles and practice of constraint programming* (pp. 565–578).

Kinable, J., Cire, A. A., & van Hoeve, W. J. (2017). Hybrid optimization methods for time-dependent sequencing problems. *European Journal of Operational Research, 259*, 887–897.

Kolen, A. W., Lenstra, J. K., Papadimitriou, C. H., & Spieksma, F. C. (2007). Interval scheduling: A survey. *Naval Research Logistics, 54*, 530–543.

Labadie, N., Mansini, R., Melechovský, J., & Wolfler Calvo, R. (2012). The team orienteering problem with time windows: An LP-based granular variable neighborhood search. *European Journal of Operational Research, 220*, 15–27.

Lawler, E. L. (1973). Optimal sequencing of a single machine subject to precedence constraints. *Management Science, 19*, 544–546.

Li, X., & Ventura, J. A. (2020). Exact algorithms for a joint order acceptance and scheduling problem. *International Journal of Production Economics, 223*, 107516.

Lin, S. W., & Ying, K. C. (2017). Increasing the total net revenue for single machine order acceptance and scheduling problems using an artificial bee colony algorithm. *Journal of the Operational Research Society, 64*, 293–311.

Liu, X., Laporte, G., Chen, Y., & He, R. (2017). An adaptive large neighborhood search metaheuristic for agile satellite scheduling with time-dependent transition time. *Computers & operations research, 86*, 41–53.

Mnich, M., & Wiese, A. (2015). Scheduling and fixed-parameter tractability. *Mathematical programming, 154*, 533–562.

Nguyen, S. (2016). A learning and optimizing system for order acceptance and scheduling. *The International Journal of Advanced Manufacturing Technology, 86*, 2021–2036.

Nobibon, F. T., & Leus, R. (2011). Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment. *Computers & Operations Research, 38*, 367–378.

Oguz, C., Sibel Salman, F., & Bilgintürk Yalçın, Z. (2010). Order acceptance and scheduling decisions in make-to-order systems. *International Journal of Production Economics, 125*, 200–211.

Pinedo, M. (2012). *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media.

Qureshi, A. G., Taniguchi, E., & Yamada, T. (2009). An exact solution approach for vehicle routing and scheduling problems with soft time windows. *Transportation Research Part E, 45*, 960–977.

Silva, Y. L. T. V., Subramanian, A., & Pessoa, A. A. (2018). Exact and heuristic algorithms for order acceptance and scheduling with sequence-dependent setup times. *Computers & operations research, 90*, 142–160.

Slotnick, S. A. (2011). Order acceptance and scheduling - a taxonomy and review. *European Journal of Operational Research, 212*, 1–11.

Spieksma, F. C. R. (1999). On the approximability of an interval scheduling problem. *Journal of Scheduling, 2*, 215–227.

Subramanian, A., & Farias, K. (2017). Efficient local search limitation strategy for single machine total weighted tardiness scheduling with sequence-dependent setup times. *Computers & Operations Research, 79*, 190–206.

Woeginger, G. J. (2003). Exact algorithms for NP-hard problems: A survey. *Lecture Notes in Computer Science, 2570*, 185–208.

Zhang, L., Lu, L., & Yuan, J. (2009). Single machine scheduling with release dates and rejection. *European Journal of Operational Research, 198*, 975–978.