Optimal Mechanism Design for a Sequencing Problem with Two-Dimensional Types

Ruben Hoeksma[1] and Marc Uetz[2]

[1] Universidad de Chile, Departamento de Ingeniería Industrial, Santiago, Chile, rubenh@dii.uchile.cl
[2] University of Twente, Department of Applied Mathematics, Enschede, The Netherlands, m.uetz@utwente.nl

Abstract. We study the design of mechanisms for a sequencing problem where the types of job-agents consist of processing times and waiting costs that are private to the jobs. In the Bayes-Nash setting, we seek to find a sequencing rule and incentive compatible payments that minimize the total expected payments that have to be made to the agents. It is known that the problem can be efficiently solved when jobs have single dimensional types. Here we address the problem with two-dimensional types. We show that the problem can be solved in polynomial time by linear programming techniques, answering an open problem formulated by Heydenreich et al. Our implementation is randomized and truthful in expectation. Remarkably, it also works when types are correlated across jobs. The main steps are a compactification of an exponential size linear programming formulation, and a convex decomposition algorithm that allows us to implement the optimal linear programming solution. In addition, by means of computational experiments we generate some new insights into the implementability in different equilibria.

1 Introduction & Contribution

This paper addresses Bayesian mechanism design for a basic sequencing problem that has been introduced by Heydenreich et al. (2008). The setting is a simple scheduling problem with private data: A number of n clients are queueing for a service, the service provider needs to compensate all clients for their waiting time, but waiting costs and service times are private to the clients. This problem is an abstraction of economic situations where clients queue for a single scarce resource, e.g., a specialized operation theater, where the information on the urgency and duration to treat each client is private, yet known probabilistically. A concrete example for the latter that motivates the present study are waiting lists for medical treatments in the Netherlands (Kenis, 2006). At the same time, the problem is the private information version of one of the most basic and classical single machine scheduling problems, namely to minimize the total weighted completion time of nonpreemptive jobs with weights w_j and processing times p_j. This problem is close to trivial from the optimization point of view, and the optimal sequence is to process the jobs in order of non-increasing ratios weight over processing time, w_j/p_j, also referred to as Smith's rule (Smith, 1956). However, once the data w_j and p_j is private the solution is less obvious.

The problem that we solve is this: There are n jobs with two-dimensional types, namely a cost per unit waiting time, w_j, and a processing time, p_j. Jobs need to be scheduled sequentially and non-preemptively, and each job requires monetary compensation for the disutility of waiting. Assuming that we are given probabilistic information on the possible types of all jobs, we seek to find a sequencing rule, along with an incentive compatible payment rule, that minimizes the total expected payments that have to be made to the jobs. Here, incentive compatibility refers to the Bayes-Nash equilibrium. As our main result we show that this optimal mechanism design problem with two-dimensional types can be solved and implemented in polynomial time. Our approach even allows the types of the jobs to be correlated. We thereby answer an open question posed by Heydenreich et al. (2008). Our solution is based on linear programming, and results in an optimal randomized mechanism.

The paper has two major technical contributions. The first is the compactification of an exponential size linear programming formulation of the mechanism design problem. This compactification yields a polynomial size linear programming relaxation. The second is an algorithm that allows to translate the solution of the linear programming relaxation into an actual implementation of the mechanism, that is, a randomization over schedules. To that end, we reduce the implementation problem to that of

computing the intersection of a line with the single machine scheduling polytope, for which we give a combinatorial $O(n^2 \log n)$ algorithm. The implementation problem is thereby solved in time $O(n^3 \log n)$. Moreover, we present some computational results based on the linear programming formulation. The primary goal of these computations is to test and validate hypotheses on the structure of solutions. Our computations, based on randomly generated instances, show that optimal mechanisms in the two-dimensional setting do *not* share several of the nice properties of the solutions to the single dimensional problem: The scheduling rules of optimal Bayes-Nash incentive compatible mechanisms are not necessarily *iia* (independent of irrelevant alternatives) and optimal Bayes-Nash mechanisms do not necessarily allow an implementation in dominant strategies. This is in contrast to the single dimensional problem, which has these properties (Duives et al., 2015).
In addition, we address a variation of the problem where job-agents are also allowed to understate their true processing requirement. For that problem, we derive a purely combinatorial solution by showing that the problem effectively reduces to a stochastic single machine scheduling problem, which is solved by the stochastic version of Smith's rule.

2 Related Work

The starting point of this paper is the open problem formulated by Heydenreich et al. (2008) who 'leave it as an open problem to identify (closed formulae for) optimal mechanisms for the 2-d case.' Here, the '2-d case' refers to the problem of computing a Bayes-Nash optimal mechanism for the mechanism design problem with two-dimensional types, where weights and processing times of the jobs are private information. The case where types are single-dimensional, and only the weights are private, can be solved efficiently along the lines of Myerson (1981). We refer to Heydenreich et al. (2008) and Duives et al. (2015) for details, and note that they also give structural insights into the case where types are two-dimensional. Yet, the computational complexity of the optimal mechanism design problem with two-dimensional types was left open.
Even though we settle the computational complexity of the problem to compute an optimal (randomized) mechanism, we do not obtain 'closed formulae' for its solution. Our results can therefore be seen in the tradition of 'automated mechanism design' as proposed e.g. by Conitzer and Sandholm (2002) and Sandholm (2003), since the design of the mechanism itself is based on linear programming.
There is abundant related work in optimal mechanism design, starting with the seminal paper by Myerson (1981). As a matter of fact, problems with single dimensional types are considered pretty well understood, and we refer to the chapter on profit maximization in mechanism design by Hartline and Karlin (2007). Algorithmic results for problems with multidimensional types have been obtained more recently (e.g. Alaei et al., 2012; Cai et al., 2012; Daskalakis and Weinberg, 2012), with a fast growing literature. We refer to a recent survey by Chawla and Sivan (2014) for an excellent overview and relevant references for recent advances in algorithmic Bayesian mechanism design. There is also a series of recent papers that provide insight into interesting anomalies in multi-dimensional mechanism design, or more specifically, multi-item auction problems (e.g. Hart and Nisan, 2014; Hart and Reny, 2015).
With respect to recent work on mechanism design with multidimensional types, our work has some methodological similarities with recent work on Bayesian mechanism design by Daskalakis and Weinberg (2012). They address a multidimensional, multi-item auction problem, and also compactify an exponential size linear program by exploiting symmetry in value distributions. Doing that, they derive approximately optimal, randomized Bayes-Nash mechanisms. Due to implicit informational externalities[3] the sequencing problem considered here cannot easily be cast in those terms. Yet, with some additional work one can observe that the techniques of Cai et al. (2012) can be applied also to the problem studied here, and then imply that a close to optimal Bayesian mechanism can be found in polynomial time. However, those techniques do not lead to an explicit, polynomial size linear programming model for the optimal mechanism design problem, which we do provide here. Moreover, our compactification does not rely on eliminating symmetries in value distributions, but rather on the elimination of what could be called 'irrelevant alternatives' for the optimal randomized mechanism.
Also, with respect to the actual implementation of the mechanism, we acknowledge a close similarity to the work of Cai et al. (2012), since also there, interim allocations need to be translated into randomizations over mechanisms, in their case VCG allocation rules. But instead of relying on a general

[3] That is, the valuation of an agent for a given solution depends on types of other agents, too.

separation oracle and the ellipsoid method, we here explicitly solve the underlying decomposition problem by giving a combinatorial algorithm. This is of independent interest both from the perspective of mechanism design and from the perspective of polyhedral combinatorics.

3 Definitions & Preliminary Results

We consider a sequencing problem with n job-agents denoted $j \in N$, each owning a job with weight w_j and processing time p_j. The jobs need to be sequenced non-preemptively on a single machine, with the interpretation that w_j is job j's individual cost for waiting one unit of time, while p_j is the time it requires to process job j. In a schedule where job j has start time s_j, the job?s individual cost for waiting s_j time units equals $w_j s_j$. The *type* of a job j is the two-dimensional vector of weight and processing time, denoted $t_j = (w_j, p_j)$. If t_j is public, the total waiting cost is well known to be minimized by sequencing the jobs in order of non increasing ratios w_j/p_j, known as Smith's rule (Smith, 1956).
In the setting we consider here, weight and processing time are private to the agent that owns the job. There is, however, a public belief about this private information, which is

- the types that jobs $1, \ldots, n$ might have are $t = (t_1, \ldots, t_n) \in T$, and T is known, and
- the probability of the jobs having types $t = (t_1, \ldots, t_n)$ is $\varphi(t)$, and also φ is known .

Hence, $T = T_1 \times \ldots \times T_n$ is the type space of all jobs, and we refer to the type of one specific job j as t_j, with $t_j \in T_j = \{t_j^1, \ldots, t_j^{m_j}\}$, where T_j denotes the set of m_j possible types for this job. Notice that we assume the type space T to be discrete. If necessary, continuous type spaces could be approximated by corresponding discretizations. Indeed, in the words of Vohra (2012), 'nothing of qualitative significance is lost in moving from a continuous to a discrete type space'.
We define $m := \sum_{j \in N} m_j$, and note that $m \geq n$. For a type $t_j \in T_j$, we let w_j and p_j be the corresponding weight and processing time, respectively. For convenience, we also use the notation $p_j(t_j)$ and $w_j(t_j)$ to denote the processing time and weight of a job with type $t_j = (w_j, p_j)$, as well as $p_j(t)$ and $w_j(t)$ to denote the processing time and weight of a job j in type vector t. In subsequent linear programming formulations, again for convenience, we sometimes add an index for the specific type i of a given job j. We then use t_j^i for that type, and often identify i with t_j^i, to avoid excessive notation.
As usual (t_j, t_{-j}) denotes a type vector where t_j is the type of job j and t_{-j} are the types of all jobs except j, with $t_{-j} \in T_{-j} := \prod_{k \neq j} T_k$. Generally, we may even allow correlation across types of different jobs. In that case, for fixed type t_j of job j, we let $\varphi(t_j) := \sum_{t_{-j}} \varphi(t_j, t_{-j})$ be the (unconditional) probability of job j having type t_j. Likewise, let $\varphi(t_{-j}) := \sum_{t_j} \varphi(t_j, t_{-j})$. Then $\varphi(t_j | t_{-j}) = \varphi(t_j, t_{-j})/\varphi(t_{-j})$ is the conditional probability for job j having type t_j, given t_{-j} is the types of all other jobs. Likewise, $\varphi(t_{-j} | t_j) = \varphi(t_j, t_{-j})/\varphi(t_j)$.
For uncorrelated type distributions, the succinct input of the problem consists of m_j types for all jobs j, hence of $m = \sum_{j \in N} m_j$ pairs of numbers $t_j = (w_j, p_j)$, with corresponding probabilities $\varphi(t_j)$. In that case, the input size of the problem is of order $\Theta(m)$, assuming that each of these numbers has size O(1). For correlated types, the input size of the problem could be as large as the size of the type space T, which may be exponential in m. In that case, to circumvent complicated reasoning about the input size of the type distribution, we will assume that it is accessed through oracle queries for any value $\varphi(\,\cdot\,)$, conditional or unconditional, in O(1) time.
We assume, like Heydenreich et al. (2008), that the mechanism designer needs to compensate the jobs for waiting by a payment. We seek to compute and implement a (direct) mechanism, consisting of a scheduling rule and a payment rule. More specifically, the mechanism assigns to any type vector $t \in T$ a vector $s(t)$ that represents the start times of all jobs in the sequence selected by the mechanism, together with a vector of compensation payments $\pi(t)$, with $\pi_j(t)$ being the payment for job j. Clearly, jobs may have an incentive to strategically misreport their true types in order to receive earlier positions in the sequence and/or higher compensation payments. The optimal mechanism that we seek is *not* welfare maximization, which is equivalent with minimizing the total waiting time. Rather, we seek a mechanism that minimizes the total payments made to the jobs. This is the equivalent to the revenue maximizing auction of Myerson (1981).

3.1 Modeling Private Processing Times

For the major part of this paper we assume, like Heydenreich et al. (2008), that job-agents can misreport their true processing times p_j, but with the restriction that only larger than the true processing times

can be reported by any job. This assumption is justified in a model where job-agents need to make sure that they receive at least their (true) required processing. That means that, effectively, their utility is independent of the actual processing they receive, as long as it is at least as large as the true p_j. Reporting a processing time *smaller* than the true processing time would result in leaving the job unfinished and this results in a zero (or any negative) utility. Hence no job-agent would ever choose this option. On the other hand, a job-agent may strategize on reporting a larger than true processing time if that increases the (expected) utility.
In addition, in Appendix A we address the alternative setting where jobs *can* report processing times smaller than their true processing time. As before, the underlying assumption is that the utility of a job-agent is independent of the actual processing time received, in the sense that a job is satisfied with the claimed processing time. We show that in this setting the optimal mechanism is the one that sequences the jobs in the order of the ratios *virtual weight* over *expected* processing time. Interestingly, this result is based on the insight that the two-dimensional mechanism design problem reduces to a single dimensional problem, yet with stochastic processing times. The optimal mechanism then follows from the fact that the stochastic single machine scheduling problem is solved by Smith's rule, with processing times replaced by expected processing times (Rothkopf, 1966).

3.2 Incentive Compatibility

For a given mechanism, we denote by Es_j^i and π_j^i the expected start time and payment for job j when he reports to be of type t_j^i, where the expectation Es_j^i is taken over all (truthful) reports of other jobs $t_{-j} \in T_{-j}$. Then the expected (quasi-linear) utility for job j with true type $t_j^i = (w_j^i, p_j^i)$ is exactly

$$\pi_j^i - w_j^i Es_j^i \,.$$

A mechanism is truthful, or more precisely *Bayes-Nash incentive compatible*, if it fulfils the following, linear constraint

$$\pi_j^i - w_j^i Es_j^i \geq \pi_j^{i'} - w_j^i Es_j^{i'} \,,$$

for all jobs j and types $t_j^i, t_j^{i'} \in T_j$, such that $p_j(t_j^i) \leq p_j(t_j^{i'})$. The constraint says that reporting types truthfully yields higher expected utility than anything else. A scheduling rule for which there exists a payment scheme so that the resulting mechanism is Bayes-Nash incentive compatible, is called Bayes-Nash *implementable*. Note that in the Bayesian setting, payments π_j^i are defined per type t_j^i of any job j, but not for each of the (in general exponentially many) vectors of types $t \in T$.
Moreover, we impose (expected) *individual rationality*, that is, the expected utility of any job should be nonnegative,

$$\pi_j^i - w_j^i Es_j^i \geq 0 \,.$$

This constraint makes sure that the optimization problem under consideration is bounded.
It is interesting to ask if a scheduling rule can even be implemented in the stronger *dominant strategy* equilibrium. Manelli and Vincent (2010) indeed show the equivalence of Bayes-Nash and dominant strategy implementations for the case of standard single unit private value auctions. In a dominant strategy equilibrium, reporting the true type maximizes the utility of a job not only in expectation but for *any* report t_{-j} of the other jobs, that is,

$$\pi_j(t_j^i, t_{-j}) - w_j^i s_j(t_j^i, t_{-j}) \geq \pi_j(t_j^{i'}, t_{-j}) - w_j^i s_j(t_j^{i'}, t_{-j})$$

for all $t_j^i, t_j^{i'} \in T_j$ and all $t_{-j} \in T_{-j}$. The latter obviously implies the former, but generally not vice versa (Gershkov et al., 2013). We comment on dominant strategy implementations in Section 6, but are mainly interested in the Bayesian setting.
For the problem considered here, it is known that implementability is equivalent to monotonicity with respect to weights:

Theorem 1 (Duives et al. (2015)). *A mechanism is Bayes-Nash implementable if and only if the expected start times Es_j^i are monotonically decreasing in the reported weight w_j^i.*

The same result holds for dominant strategy implementability, but then the start times $s_j(t_j^i, t_{-j})$ need to be monotonically decreasing in the reported weight w_j^i, for all $t_{-j} \in T_{-j}$. This is a standard result in single-dimensional mechanism design, see for instance the introductory text by Nisan (2007), but

it is also true for the two-dimensional problem considered here; see Duives et al. (2015, Theorems 1 and 6). With respect to optimal mechanisms, for the special case where only weights w_j are private and processing times p_j are known, the Bayes-Nash optimal mechanism has a simple combinatorial structure. It is Smith's rule with respect to *virtual* instead of the original weights, i.e., jobs are sequenced in non-increasing order of the ratios virtual weight over processing time (Duives et al., 2015).

4 Problem Formulation & Linear Relaxation

We first set up an integer linear programming formulation that describes the problem of finding an optimal Bayes-Nash incentive compatible and individual rational mechanism. The starting point is an integer linear programming formulation for the scheduling polytope in terms of so-called linear ordering variables, $d_{kj}(t)$, with intended meaning

$$d_{kj}(t) = \begin{cases} 1 & \text{if for type vector } t \text{ we use a schedule where job } k \text{ precedes job } j\,, \\ 0 & \text{otherwise}\ . \end{cases}$$

See also Dyer and Wolsey (1990). The following constraints ensure that the linear ordering variables indeed describe a linear ordering:

$$d_{jj}(t) = 0 \qquad \forall j, t \tag{1}$$
$$d_{kj}(t) + d_{jk}(t) = 1 \qquad \forall j, k, t\ j \neq k \tag{2}$$
$$d_{jk}(t) + d_{kl}(t) \leq 1 + d_{jl}(t) \qquad \forall j, k, l, t \tag{3}$$
$$d_{jk}(t) \in \{0, 1\} \qquad \forall j, k, t\,. \tag{4}$$

Here (3) is the triangle inequality. In terms of these linear ordering variables, for any given vector of types t, the corresponding start times of jobs in the sequence corresponding to that linear ordering are linearly expressed as

$$s_j(t) = \sum_{k \in N} d_{kj}(t) p_k(t)\ \ , \tag{5}$$

recalling that $p_j(t)$ denotes the processing time of job j in type profile t. Observe that, for a given type vector t, the vectors $s(t)$ given by (1)-(5) are vectors of start times of the jobs. These are the vertices of the well known single machine scheduling polytope (Dyer and Wolsey, 1990; Queyranne, 1993), with the only difference that we consider start instead of completion times. The polytope is exactly the convex hull of all vectors of start times of all $n!$ job sequences. Here, it will be denoted by $Q(t)$. It is well known that the scheduling polytope is a (contra)polymatroid, which is easily verified by the transformation to $x_j(t) = p_j(t)s_j$ for all $j \in N$. Note that both linear optimization over $Q(t)$, as well as the separation problem for $Q(t)$ can be solved in time $\mathrm{O}(n^2)$ (Edmonds, 1971; Queyranne, 1993).
As it will be important for what follows, we next recall a well known linear description of the single machine scheduling polytope due to Queyranne (1993). To that end, recall that the variables $(s_1, \ldots, s_n)$ denote the start times of jobs. We know from Queyranne (1993) that the facial description of the polytope $Q(t)$ is given by the following set of inequalities

$$\sum_{j \in K} p_j(t) s_j \geq \frac{1}{2} \left(\sum_{j \in K} p_j(t) \right)^2 - \frac{1}{2} \sum_{j \in K} p_j(t)^2 \qquad \forall K \subset N \tag{6}$$

$$\sum_{j \in N} p_j(t) s_j = \frac{1}{2} \left(\sum_{j \in N} p_j(t) \right)^2 - \frac{1}{2} \sum_{j \in N} p_j(t)^2\ \ . \tag{7}$$

The last equality excludes schedules with idle time. The vertices of $Q(t)$ are exactly the vectors of start times of all $n!$ job sequences. Note that $Q(t)$ is $(n-1)$-dimensional. Therefore, any (inner) point of $Q(t)$ represents feasible expected start times of a randomization over (at most n) schedules, by Carathéodory's theorem.

Using linear ordering variables now yields the following mixed integer linear programming formulation of the optimal Bayesian mechanism design problem. Here we use the shorthand notation $\varphi_j^i := \varphi(t_j^i)$ for the probability of job j having type t_j^i. Also, by $t \ni t_j^i$ we denote all type vectors $t = (t_j^i, t_{-j}) \in T$.

$$\min \sum_{j \in N} \sum_{t_j^i \in T_j} \varphi_j^i \pi_j^i \tag{8}$$

$$\text{s.t.} \quad \pi_j^i \geq w_j^i E s_j^i \qquad \forall j, i \tag{9}$$

$$\pi_j^i \geq \pi_j^{i'} - w_j^i (E s_j^{i'} - E s_j^i) \qquad \forall j, i, i',\ p_j(t_j^{i'}) \geq p_j(t_j^i) \tag{10}$$

$$E s_j^i = \sum_{t \ni t_j^i} \varphi(t_{-j} \mid t_j^i) s_j(t_j^i, t_{-j}) \qquad \forall j, i \tag{11}$$

$$s_j(t) = \sum_{k \in N} d_{kj}(t) p_k(t) \qquad \forall j, t \tag{12}$$

$$d_{jj}(t) = 0 \qquad \forall j, t \tag{13}$$

$$d_{kj}(t) + d_{jk}(t) = 1 \qquad \forall j, k, t\ j \neq k \tag{14}$$

$$d_{jk}(t) \geq 0 \qquad \forall j, k, t \tag{15}$$

$$d_{jk}(t) + d_{kl}(t) \leq 1 + d_{jl}(t) \qquad \forall j, k, l, t \tag{16}$$

$$d_{jk}(t) \in \{0, 1\} \qquad \forall j, k, t\ . \tag{17}$$

Note that the only free variables are linear ordering variables $d_{jk}(t)$ as well as payments π_j^i. The variables $s_j(t)$ and $E s_j^i$ for start times and expected start times, respectively, can be eliminated. The objective (8) is the total expected payment. Constraints (9) and (10) are the individual rationality and incentive compatibility constraints: (9) requires the expected payment to at least match the expected cost of waiting when the type is t_j^i, and (10) makes sure that the expected utility is maximized when reporting truthfully. The values $E s_j^i$ are also referred to as an *interim schedule*. Indeed, $E s_j^i$ is the expected start time of job j given it has (reported) type i. Observe that the number of variables $d_{jk}(t)$ equals $n^2 \cdot |T|$, which due to the size of the type space T may be exponential in the input size of the problem.

Recall that the vertices of $Q(t)$ are the solutions $s(t)$ of (12)-(17). Moreover, a vector of start times $s(t)$ satisfies (12)-(15) if and only if $s(t) \in Q(t)$; see for instance Queyranne and Schulz (1994, Thm. 4.1). More specifically, via (12), the scheduling polytope $Q(t)$ is an affine image of both the linear ordering polytope (13)-(16) and its relaxation (13)-(15). This important observation is crucial for what follows, as it allows us to work with the relaxation (13)-(15) instead of (13)-(16).

4.1 Relaxation & Compactification

A linear relaxation of the optimal mechanism design problem (8)-(17) is obtained by dropping the last two sets of constraints (16) and (17). By moving from the ILP formulation to its LP relaxation, we in fact move from deterministic scheduling rules to randomized ones, which follows from our previous discussion about the equivalence of (12)-(15) and (6)-(7), as well as the fact that the scheduling polytope $Q(t)$ is an affine image of the relaxation (13)-(15) via (12).

In what follows we also combine (11) and (12) into just one constraint. Note that the single machine scheduling polytope is described exactly by (12)-(15) and indeed the triangle inequality, (16), is redun-

dant (Queyranne, 1993). This gives us the following formulation for the linear relaxation.

$$\min \sum_{j\in N} \sum_{t^i_j \in T_j} \varphi^i_j \pi^i_j \tag{18}$$

$$\text{s.t.} \quad \pi^i_j \geq w^i_j Es^i_j \qquad \forall j,i \tag{19}$$

$$\pi^i_j \geq \pi^{i'}_j - w^i_j (Es^{i'}_j - Es^i_j) \qquad \forall j,i,i',\ p_j(t^{i'}_j) \geq p_j(t^i_j) \tag{20}$$

$$Es^i_j = \sum_{t \ni t^i_j} \varphi(t_{-j} \mid t^i_j) \sum_{k\in N} d_{kj}(t^i_j, t_{-j}) p_k(t_{-j}) \qquad \forall j,i \tag{21}$$

$$d_{jj}(t) = 0 \qquad \forall j,t \tag{22}$$

$$d_{kj}(t) + d_{jk}(t) = 1 \qquad \forall j,k,t,\ k \neq j \tag{23}$$

$$d_{kj}(t) \geq 0 \qquad \forall j,k,t\ . \tag{24}$$

We now focus on the projection to variables Es^i_j, that is, vectors $Es \in \mathbb{R}^m$ that satisfy (21)-(24). These are interim schedules in the linear relaxation. Let us refer to this projection as the *relaxed interim scheduling polytope*. Notice that, even though it is a linear relaxation, (21)-(24) is still an exponential size formulation in general, as it depends on the size of the type space T. The crucial insight is that in the linear relaxation, this exponential size formulation is not necessary. Instead of using $d_{kj}(t)$ where $t \in T$, we propose an *LP compactification* by restricting to variables

$$d_{kj}(t_j, t_k)\,,$$

where t_j and t_k are the types of jobs j and k, respectively. Note what this means: The variable $d_{kj}(\,\cdot\,)$ that describes the linear order of two jobs j and k in the problem formulation, now only depends on the types t_j and t_k of the jobs j and k, and no longer on the whole type vector $t = (t_1, \ldots, t_n)$.
This restriction reduces the number of d_{kj}-variables to $\mathrm{O}(m^2)$, yielding a formulation of size polynomial in m. Note that this is a polynomial size formulation. Doing so, we obtain the following formulation for the interim schedule.

$$Es^i_j = \sum_{k\in N} \sum_{t_k \in T_k} \varphi(t_k \mid t^i_j) d_{kj}(t^i_j, t_k) p_k(t_k) \qquad \forall j,i \tag{25}$$

$$d_{jj}(t_j, t_j) = 0 \qquad \forall j,t_j \tag{26}$$

$$d_{kj}(t_k, t_j) + d_{jk}(t_j, t_k) = 1 \qquad \forall j,k,t_j,t_k,\ k \neq j \tag{27}$$

$$d_{kj}(t_k, t_j) \geq 0 \qquad \forall j,k,t_j,t_k\ . \tag{28}$$

The following lemma is the core technical insight of the main result in this paper.

Lemma 2. *The relaxed interim scheduling polytope defined by* (21)-(24) *can be equivalently described by* (25)-(28)*.*

Proof. Proof. Let P be the projection of (21)-(24) to variables Es^i_j, and P' be the projection of (25)-(28) to variables Es^i_j. It is pretty straightforward to verify that if $Es \in P'$, then $Es \in P$, simply by letting $d_{kj}(t) := d_{kj}(t_j, t_k)$ for all $t \ni t_j, t_k$. Here $t \ni t_j, t_k$ denotes all type vectors t in which jobs j and k have types t_j and t_k, respectively.
The more interesting step is to show is that if $Es \in P$, then $Es \in P'$. So let $Es \in P$ with corresponding $d_{kj}(t)$. Now define

$$d_{kj}(t_j, t_k) = \sum_{t \ni t_j, t_k} \frac{\varphi(t)}{\varphi(t_k, t_j)} d_{kj}(t)\ ,$$

as the weighted average of the values $d_{kj}(t)$ for those type vectors in which jobs j and k have types t_j and t_k, respectively. Here, following earlier notation we let $\varphi(t_k, t_j) := \sum_{t \ni t_k, t_i} \varphi(t)$ be the probability of jobs k and j having types t_k and t_j, respectively. In the uncorrelated case, note that $\varphi(t_k, t_j) = \varphi(t_k)\varphi(t_j)$. In either case, the so defined values $d_{kj}(t_j, t_k)$ clearly satisfy (26)-(28). Moreover, for all

$j \in N$ and fixed $t_j^i \in T_j$, we get

$$\begin{aligned}
Es_j^i &= \sum_{t \ni t_j^i} \varphi(t_{-j} \mid t_j^i) \sum_{k \in N} d_{kj}(t) p_k(t) \\
&= \sum_{t \ni t_j^i} \frac{\varphi(t)}{\varphi(t_j^i)} \sum_{k \in N} d_{kj}(t) p_k(t) \\
&= \sum_{k \in N} \sum_{t \ni t_j^i} \frac{\varphi(t)}{\varphi(t_j^i)} d_{kj}(t) p_k(t) \\
&= \sum_{k \in N} \sum_{t_k \in T_k} \varphi(t_k \mid t_j^i) \sum_{t \ni t_k, t_j^i} \frac{\varphi(t)}{\varphi(t_j^i)\varphi(t_k \mid t_j^i)} d_{kj}(t) p_k(t_k) \\
&= \sum_{k \in N} \sum_{t_k \in T_k} \varphi(t_k \mid t_j^i) d_{kj}(t_j^i, t_k) p_k(t_k) \ ,
\end{aligned}$$

which is exactly the right hand side of (25).

We conclude with the following theorem.

Theorem 3. *Computing an optimal interim schedule $Es \in \mathbb{R}^m$ together with optimal payments $\pi \in \mathbb{R}^n$ for the Bayesian mechanism design problem can be done in time polynomial in the input size of the problem by solving the compactified linear program* (18)-(20)*,* (25)-(28)*.*

Proof. Proof. In the case of uncorrelated types, the input size of the problem is $\Theta(m)$. The linear formulation (18)-(20) together with (25)-(28) has $O(m^2)$ variables and $O(m^2)$ constraints. Hence, this linear program can be solved in time polynomial in the input size. For the correlated case, the formulation is polynomial size, too: all we need to set up the formulation is that the $O(m^2)$ values $\varphi(t_k \mid t_j^i)$ can be computed in polynomial time.

Theorem 3 tells us that we can compute optimal payments and an interim schedule in polynomial time. However, an important issue remains, namely the actual implementation of the mechanism, and the question if we did not loose anything on the way by reducing the number of variables. Before we proceed to show how the optimal LP solution can be implemented in Section 5, we briefly discuss the LP relaxation.

4.2 The Compactification and the Constraint Matrix

We consider a relaxation of the linear ordering polytope by dropping triangle and integrality constraints. Also, we have reduced the number of variables from a potentially exponential number to a polynomial number. It seems that thereby we are reducing the (number of) feasible mechanisms, because the variables $d_{kj}(t_j, t_k)$ only depend on the types of jobs j and k, while $d_{kj}(t)$ depends on the whole type vector t. For deterministic mechanisms, this independence is also known as *independence of irrelevant alternatives*, or *iia-property*.

Definition 4 (*iia*). *A deterministic scheduling rule is* independent of irrelevant alternatives *or iia if the relative order of two jobs does not depend on anything but the types of those two jobs, that is $d_{kj}(t) = d_{kj}(t_j, t_k)$ for all $t \ni t_j, t_k$. We call a mechanism for which the scheduling rule is iia an iia*-mechanism*.*

Lemma 2 shows that the compactification is no loss of generality as far as the linear relaxation is concerned. With this in mind, a possible interpretation of Lemma 2 would be that the restriction to *iia*-mechanisms can be done without loss of generality once randomization is allowed. However, we did not define what a randomized *iia*-mechanism is, and this is not so straightforward. For example, it follows from Theorem 9 below that the optimal randomized mechanism cannot be represented as a lottery over deterministic *iia*-mechanisms. One reason for this is that the variables d_{kj} in the relaxation in general cannot be interpreted as the probability of job k preceding job j: By definition of the relaxation, neither the vector of variables $d_{kj}(t_j, t_k)$ nor $d_{kj}(t)$ necessarily lie in the linear ordering polytope; see e.g.

Fishburn (1992). Hence, the interpretation of the optimal randomized mechanism as an *iia*-mechanism is problematic.

At this point it is important to realize that the restriction to *iia*-mechanisms *is* a loss of generality for the deterministic optimal mechanism design problem (8)-(17): Duives et al. (2015) give an instance that shows the existence of an optimality gap in general. That is, there exist instances where the optimal deterministic *iia*-mechanism has higher total expected payments than the optimal deterministic mechanism; see also Theorem 7 below.

The underlying reason for this effect can be found by studying the structure of the constraint matrix for both problems: Recall that in the compacified linear programming relaxation, the triangle inequality, (16), is redundant. Therefore it suffices to use inequalities (21)-(24). The resulting constraint matrix, (21)–(24) is block diagonal, with one block for each pair of jobs and types. Furthermore, variables $d_{jk}(t)$ only play a role in the expected start time of jobs j and k, but no other job. Therefore it suffices if these decision variables only depend on the types of those two jobs. While all this is true for the linear relaxation, the triangle inequality breaks the block diagonal structure of the constraint matrix, and therefore *is* necessary for the integer linear program of deterministic mechanisms. As a result, for deterministic mechanisms, the decision variables need to be dependent on the whole type vector t, in general.

5 Implementation of the Optimal Mechanism

As a consequence of the preceding discussion, the solution to the compactified linear programming relaxation does not yet qualify as a solution to the Bayesian mechanism design problem, because the interim schedule Es does not have an interpretation as a randomization over schedules. In order to actually implement the mechanism, we therefore still need to compute, for any reported type vector t, the corresponding (randomized) schedule $s(t)$, so that $Es = E_T[s(t)]$. We here show that projecting to the space of start time vectors allows us to do that.

First, observe that for a given solution of the LP relaxation and any fixed type vector $t = (t_1, \ldots, t_n)$ we have values $d_{jk}(t_j, t_k)$, for each pair of jobs j and k. From these we can compute a corresponding vector of start times $s(t)$ by

$$s_j(t) = \sum_{k \in N} d_{kj}(t_j, t_k) p_k(t_k) \text{ for all } j \in J \ .$$

Now $s(t)$ is a point in the scheduling polytope $Q(t)$ defined in (6) and (7) and the dimension of $Q(t)$ is $n - 1$ (if there are no jobs with zero processing times, at least). It follows from Caratheodory's Theorem that $s(t)$ can be expressed as the convex combination of at most n vertices of $Q(t)$, that is, job sequences. This will be our desired solution to the Bayesian mechanism design problem, as this allows us to interpret the LP solution, for each reported vector of types $t \in T$, as a lottery over at most n job sequences.

For the subsequent discussion, for notational convenience, we drop the dependence on the type vector t. In order to compute a decomposition à la Caratheodory we use a well known approach (Grötschel et al., 1988, Thm. 6.5.11): Given some point $s \in Q$, pick an arbitrary vertex v of Q, and compute the point $s' \in Q$ where the half-line through v and s leaves Q. This point lies on a face(t) of Q, and we can recurse on that face(t). We call this the *GLS method*. One iteration of the GLS method is illustrated in Figure 1.

To efficiently use the GLS method, we only need a way to efficiently compute the intersection point s' and a facet f' on which it lies. For general polymatroids, this can be done with an algorithm described by Fonlupt and Skoda (2009). For the scheduling polytope, a direct application of their result leads to an algorithm that runs in time $O(n^8)$.

But since we here deal with the single machine scheduling polytope and not with general polymatroids, we can substantially improve on that. This improvement rests on the following theorem.

Theorem 5. *Let Q be the scheduling polytope in start times induced by the vector of processing times $p \in \mathbb{R}^n_{>0}$. For given $x, y \in \mathbb{R}^n$, $y \neq 0$, the computation of the intersection of a line $L = \{x + \lambda y \mid \lambda \in \mathbb{R}\}$ with Q together with the facets of Q, which intersect with L, can be done in time $O(n^2 \log n)$.*

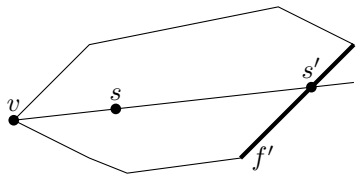

Fig. 1. Illustration of one iteration of the GLS method. The point s is a convex combination of the vertex v and the new point s', which lies on a lower-dimensional face of the polytope, f'.

The algorithm and proof of the theorem is given in Appendix B. Combined with the GLS method, for any type vector t and the corresponding vector of start time $s(t)$, we can therefore compute a representation of $s(t)$ as a convex combination of at most n job sequences in computation time $\mathrm{O}(n^3 \log n)$.

Theorem 6. *A point s in the single machine scheduling polytope Q can be decomposed into the convex combination of at most n vertices (equivalently, permutation schedules) of Q in time $O(n^3 \log n)$.*

Proof. Proof. Using the GLS method starting with $s = s^0$, the line intersection algorithm computes an intersection point s^1, and at the same time a facet of the scheduling polytope Q on which this intersection point lies. That facet is represented by some set $K_1 \subset N$ for which inequalities (6) are tight. Note that these are schedules in which all jobs in K_1 are processed before all jobs in $N \setminus K_1$. To iterate the procedure on the lower dimensional face on which s^1 lies, the next vertex v^1 can be chosen as the schedule induced by the sequence σ given by the order of the elements of s^1, say $s^1_{\sigma(1)} \leq \cdots \leq s^1_{\sigma(n)}$. That takes $\mathrm{O}(n \log n)$ time. Note that, as both v^1 and s^1 lie on the facet defined by K_1, the next iteration must yield a facet K_2 so that either $K_2 \subset K_1$ or $K_1 \subset K_2$. In other words, the algorithm produces a sequence of nested sets. Indeed, Queyranne (1993) showed that every $(n-k)$-dimensional face of Q corresponds one-to-one with an ordered partition of N into k sets, that is a tuple $(N_1, \ldots, N_k)$ with $N_i \cap N_j = \emptyset$ for all $i \neq j$, $i, j \in \{1, \ldots, k\}$, and $\bigcup_{i=1}^{k} N_i = N$. The interpretation is that inequalities (6) are tight for all k nested sets $K_i := N_1 \cup \ldots \cup N_i$, $i = 1, \ldots, k$. Indeed, the intersection point of the kth iteration lies exactly on the $(n-k)$-dimensional face defined by the ordered partition that is obtained from the nested sets $K_1, \ldots, K_k$. Now since the dimension is n, and each iteration takes $\mathrm{O}(n^2 \log n)$ time by Theorem 5, the claimed computation time follows.

We note that a point in the scheduling polytope can in fact be decomposed into a convex combination of at most n vertices in time $\mathrm{O}(n^2)$ using another algorithm that we recently found (Hoeksma et al., 2014). However in contrast to what we describe here, that algorithm does not yield the intersection of the scheduling polytope with a line. Therefore, we believe that the result presented in Theorem 5 is of independent interest, even though a faster decomposition algorithm is possible.

6 Computational Results

We have implemented the (integer) linear programming model discussed in this paper. The main purpose for the implementation was to verify a number of conjectures about the relations between implementations in different equilibria, specifically Bayes-Nash versus dominant strategy, as well as the *iia*-property. To that end, we generated and tested the implementations on randomly generated instances, the essence of which is presented here.

6.1 Bayes-Nash and dominant strategy equilibria and the *iia*-property

The following two instances are an outcome of our experiments and encompass some new, theoretical insights.

Instance 1 *Four jobs with the following type spaces and corresponding probabilities:*

Job 1	$w=6$	$w=7$	$w=10$
$p=2$	0.3312	0.3456	0.0432
$p=7$	0.1288	0.1344	0.0168

,

Job 2	$w=5$	$w=8$
$p=4$	0.0344	0.8256
$p=8$	0.0056	0.1344

,

Job 3	$w=3$	$w=10$
$p=8$	0.3825	0.1275
$p=10$	0.3675	0.1225

,

Job 4	$w=3$	$w=8$
$p=1$	0.2583	0.3717
$p=6$	0.1517	0.2183

.

Instance 2 *Three jobs with the following type spaces and corresponding probabilities:*

Job 1	$w=2$
$p=1$	1

,

Job 2	$w=9$
$p=8$	1

,

Job 3	$w=1$	$w=3$	$w=5$
$p=5$	0.24	0.02	0.16
$p=7$	0.24	0.24	0.10

.

We summarize our computational experiments in the following theorems.

Theorem 7. *Optimal deterministic mechanisms for either Bayes-Nash or dominant strategy implementations, in general do not satisfy the iia-condition.*

Proof. Proof.Duives et al. (2015) use Instance 2 to prove this theorem for optimal Bayes-Nash mechanisms. Indeed, Instance 1 shows the same: The optimal deterministic Bayes-Nash mechanism has a total expected cost of 128.5195. When enforcing the mechanism to be *iia*, the total expected cost becomes 128.5697. Moreover, the optimal deterministic dominant strategy mechanism has a total expected cost of 128.6151, while that becomes 128.6946 if the mechanism is forced to be *iia*.

Corollary 8. *The optimal deterministic Bayes-Nash mechanism is, both in the iia and the non-iia case, generally not implementable in dominant strategies.*

Theorem 9. *Randomized Bayes-Nash mechanisms perform better than deterministic Bayes-Nash mechanisms in terms of minimal total expected payment. In particular, a randomized Bayes-Nash implementation cannot, in general, be decomposed into the convex combination of deterministic Bayes-Nash implementations.*

Proof. Proof. Instance 2 has an optimal deterministic Bayes-Nash mechanism with total expected cost 45.0, while the optimal randomized Bayes-Nash mechanism has total expected cost 44.74625.

7 Concluding Remarks

Our solution is randomized and truthful in expectation. The computational complexity to find an optimal *deterministic* mechanism remains open. It is not even clear if the corresponding decision problem is in NP.
Another, interesting future path is to analyze the worst-case gaps between the different types of implementations that we studied (*iia* versus non-*iia* for example).
Moreover, it would be interesting to get qualitative information from the linear programming approach that goes beyond the statements made in Section 6. While we believe it is interesting to get polyhedral explanations for phenomena in the design of mechanisms, along the lines of Section 4.2, it could also work the other way around. Namely, one might be able to derive qualitative properties of mechanisms via structural properties of the underlying mathematical programs.

Acknowledgements. We thank Maurice Queyranne for pointing us to the paper by Yasutake et al. (2011), and Walter Kern, Marc Pfetsch, Rudolf Müller, and Gergely Csapó for helpful discussions. We also thank the associate editor and the anonymous referees for their thoughtful comments that greatly helped in improving the paper. In particular, one of the referees pointed us to the fact that our approach also works for correlated types. The first author was partially supported by Millennium Nucleus Information and Coordination in Networks ICM/FIC RC130003. An extended abstract with parts of this paper appeared in the proceedings of IPCO 2013 (Hoeksma and Uetz, 2013).

A When Processing Times Don't Matter

In none of the technical proofs of the paper it was actually necessary that the reported processing times are at least as large as the true processing times. Here we address the case where this is allowed. The first thing to note is that once this requirement is relaxed, there is a flaw in the model that needs to be fixed. Namely, if a job would do this, we cannot ensure that the proposed allocation is feasible in the sense that each job receives the required processing: The mechanism designer would not even know the true required processing of any job. As already explained in Section 3.1, the natural adaptation of the model if jobs are also allowed to report smaller than true processing times would be to assume that jobs just receive the amount of processing they claim. This boils down to saying that the utility of a job does not depend on the allocated processing, as long as it is at least as large as the claimed processing time.

In this section we show that for this model, the optimal Bayesian mechanism is very simple. Indeed, we will show how the problem reduces to a single-dimensional mechanism design problem, which then boils down to the stochastic single machine scheduling problem. The latter can be solved by the stochastic version of Smith's rule, namely to process the jobs in non-increasing order of (virtual) weight over *expected* processing time. The optimality of that version of Smith's rule in the stochastic single machine setting is well known (Rothkopf, 1966).

In order to describe the optimal mechanism for this case we need to recall some of the definitions of Duives et al. (2015). For each job j, we compute *virtual weights* as follows. Let W_j and P_j denote all the weights and processing times, respectively, of job j in all type vectors $t_j^1, \dots, t_j^{m_j}$. For notational convenience assume the different weights are indexed according to $w_j^1 \leq \dots \leq w_j^{m_j}$. Then the virtual weights are $\overline{w}_j^1 := w_j^1$ and

$$\overline{w}_j^i := w_j^i + (w_j^i - w_j^{i-1}) \frac{\sum_{k=1}^{i-1} \varphi(w_j^k)}{\varphi_j(w_j^i)} \quad \text{for } i = 2, \dots, m_j \,,$$

where $\varphi(w_j^i) := \sum_{t \ni w_j^i} \varphi(t)$ denotes the probability that job j has a type with weight w_j^i. Moreover, for some fixed weight $w_j^i \in W_j$, denote by $P_j(w_j^i) \subseteq P_j$ the processing times of job j conditioned on the weight being w_j^i. Here, the intended meaning is that $P_j(w_j^i)$ are the processing times in job j's types of the form $t_j^i = (w_j^i, \cdot\,) \in T_j$. Then let

$$Ep_j(w_j^i) := \frac{\sum_{p_j \in P_j(w_j^i)} \varphi(w_j^i, p_j) p_j}{\varphi(w_j^i)}$$

be the expected processing time of a job j conditioned on its weight being w_j^i.

Now, suppose the jobs report types $t_1^i, \dots, t_n^i$ with weights $w_1^i, \dots, w_n^i$. Then we claim that the optimal Bayesian mechanism is ordering the jobs non-increasing in the ratio of virtual weight over expected (conditional) processing time,

$$\frac{\overline{w}_j^i}{Ep_j(w_j^i)} \,.$$

The corresponding minimal payment scheme that implements this scheduling rule can be efficiently computed by shortest path computations in the underlying type graphs, as has been shown by Duives et al. (2015, Lemma 1).

Note that reporting only weights would be sufficient since both the expected processing time and the virtual weights do not depend on the actual (reported) processing time.

Theorem 10. *The Bayesian mechanism design problem where agents are allowed to report smaller than true processing times has an optimal solution where the scheduling rule is to sequence the jobs in non-increasing ratios of virtual weight $\overline{w}_j^i$ over (conditional) expected processing time $Ep_j(w_j^i)$.*

Proof. Proof. Say (Es, π) is the interim schedule and payment scheme of an optimal Bayesian mechanism. Abusing notation a little bit, let us denote by k and ℓ two types of a job j so that $t_j^k = (w_j^i, p_j^k)$ and $t_j^\ell = (w_j^i, p_j^\ell)$. That is, k and ℓ denote two types of a job j with the same weight w_j^i but with different processing times. We have by incentive compatibility that

$$\pi_j^k - w_j^i Es_j^k \geq \pi_j^\ell - w_j^i Es_j^\ell \qquad \text{for all such } t_j^k, t_j^\ell \in T_j \;\; .$$

Since this inequality holds also with k and ℓ exchanged, we conclude

$$\pi_j^k - w_j^i Es_j^k = \pi_j^\ell - w_j^i Es_j^\ell \qquad \text{for all such } t_j^k, t_j^\ell \in T_j \ .$$

In other words, Bayes-Nash incentive compatibility implies that a job j receives the same expected utility for a given reported weight w_j^i, irrespective of the reported processing time. Note that this is not necessarily the case once we forbid reporting a smaller than true processing time, since then only one of the two inequalities is present.

But we can even say more, namely, we next argue that in the optimal mechanism, even the expected start time is independent of the reported processing time. To that end, we take a look at the payments. Consider the Bayes-Nash incentive compatibility constraints between types with different weights. We know that an implementable scheduling rule must satisfy monotonicity with respect to weights. This follows from Theorem 1, which continues to hold also for the problem where jobs may understate their true processing time. (This is not difficult to see when considering the underlying type graphs.) In particular it therefore holds that

$$\min_{p_j \in P_j(w_j^i)} Es_j(w_j^i, p_j) \geq \max_{p_j \in P_j(w_j^k)} Es_j(w_j^k, p_j) \qquad \forall w_j^i < w_j^k \ . \tag{29}$$

In words, a larger weight always yields a smaller expected start time, and this is true irrespective of the reported processing time.

We next want to express minimal payments for any given implementable scheduling rule. To that end, recall from Duives et al. (2015) that, for any implementable scheduling rule, the minimal payment for reporting type $t_j^i = (w_j^i, p_j^i)$ can be computed by a shortest path calculation in the type graph. To keep the presentation brief, we refer to (Duives et al., 2015) for further details and conclude that the incentive compatible payments fulfill the following set of inequalities, and the minimal payment for reporting type (w_j^i, p_j^i) is exactly obtained by the maximum of the right hand side in

$$\pi_j(w_j^i, p_j^i) \geq \left[\sum_{k=i}^{m_j - 1} w_j^k \left(Es_j(w_j^k, p_j^k) - Es_j(w_j^{k+1}, p_j^{k+1}) \right) \right] + w_j^{m_j} Es_j(w_j^{m_j}, p_j^{m_j}) \ . \tag{30}$$

Here, the sequence of non-decreasing weights $w_j^i \leq w_j^{i+1} \leq \cdots \leq w_j^{m_j}$ is fixed and exhaustive. Indeed, for the problem considered here the only flexibility in expression (30) lies in choosing, for each $k > i$, some $p_j^k \in P_j(w_j^k)$. (This exactly corresponds to choosing different paths in the type graph.)

Now observe that each term $Es_j(w_j^k, p_j^k)$ appears in (30) with as coefficient $(w_j^k - w_j^{k-1}) Es_j(w_j^k, p_j^k) \geq 0$ and hence the maximum in the right hand side of (30) is attained when, for all $k > i$, p_j^k equals

$$p_j^k = \operatorname*{arg\,max}_{p_j \in P_j(w_j^k)} Es_j(w_j^k, p_j) \, .$$

Therefore the payments of the optimal Bayesian mechanism are minimal whenever the expression $\max_{p_j \in P_j(w_j^k)} Es_j(w_j^k, p_j)$ is minimal, for each weight w_j^k. Here, note that the feasibility of this independent choice for each k is guaranteed by (29). However this means that the mechanism is optimal only if $Es_j(w_j^k, p_j)$ is *the same* for all $p_j \in P_j(w_j^k)$. In other words, for each fixed weight w_j^k the mechanism assigns the same expected start times for all $p_j \in P_j(w_j^k)$, and by incentive compatibility also the same payment for all $p_j \in P_j(w_j^k)$.

This means that, for given weight w_j^i the (expected) start time of a job must be independent of its actual processing time. Therefore, the expected completion time of a job with weight w_j^i equals

$$Es_j(w_j^i) + Ep_j(w_j^i) \, .$$

In other words, we are effectively left with a single-parameter Bayesian mechanism design problem, where the processing time for a job with weight w_j^i equals $Ep_j(w_j^i)$.

Hence with no loss of generality we can define a scheduling rule and payment rule that only depends on the vector of reported weights of the jobs. We can now use (30), together with the definition of virtual weights given above, to write the minimal total expected payment of any implementable scheduling rule as

$$\sum_{j \in J} \sum_{w_j \in W_j} \varphi(w_j) \overline{w}_j Es_j(w_j) \, .$$

For a formal proof of the latter, refer to the appendix of Duives et al. (2015), where we only need to replace processing times p_j by expected (conditional) processing times $Ep_j(w_j^i)$. Now we have

$$\sum_{j\in J}\sum_{w_j\in W_j}\varphi(w_j)\overline{w}_j Es_j(w_j)=\sum_{j\in J}\sum_{w_j\in W_j}\sum_{w\ni w_j}\varphi(w)\overline{w}_j Es_j(w_j)=\sum_{w\in W}\varphi(w)\sum_{j\in J}\overline{w}_j Es_j(w_j)\,, \qquad (31)$$

where $w=(w_1,\ldots,w_n)$ denotes a vector of weights for all jobs, and W is the type space of all weight profiles. Therefore, conditioned on a given vector of reported weights $w=(w_1,\ldots,w_n)$, the term $\sum_{j\in J}\overline{w}_j Es_j(w_j)$ that appears in the right hand side of (31) is minimized by sequencing the jobs in the order of virtual weight $\overline{w}_j$ over conditional expected processing times $Ep_j(w_j)$. Given that we are not allowed to make the sequencing dependent on the actual reported processing times, this is indeed the best we can do, and optimality of this sequencing rule follows from a paper by Rothkopf (1966) on sequencing with random service times.
We finally note that the final argument implicitly requires the ratios $\overline{w}_j/Ep_j(w_j)$ to be monotonically increasing in w_j, as otherwise the scheduling rule is not monotone and thus not implementable. This technical issue, however, is well known to be fixable by a standard procedure known as *ironing*. We do not go into the technical details here, but refer to Myerson (1981) or Vohra (2011).

B An $O(n^2 \log n)$ Line Intersection Algorithm

We here give the proof of Theorem 5. First we give a simple argument that immediately leads to an $O(n^4)$ time bound. To start with, we (re)state a lemma that directly follows from Queyranne (1993); it shows that the separation problem for the scheduling polytope can in fact be solved by sorting. For convenience of notation let us define

$$g(K):=\frac{1}{2}\left(\sum_{j\in K}p_j\right)^2-\frac{1}{2}\sum_{j\in K}{p_j}^2$$

to be the right-hand-side of (6).

Lemma 11. *Let s be a given vector sorted such that $s_1\le s_2\le\ldots\le s_n$. Then $s\in Q$ if and only if $\sum_{j\in N}p_j s_j=g(N)$ and $\sum_{j\in K}p_j s_j\ge g(K)$ holds for all $K=\{1,\ldots,k\}$, $k=1,\ldots,n$. In particular, if there is a set $K\subseteq N$ that violates* (6)*, then there is a $k\in\{1,\ldots,n\}$ such that the set $K=\{1,\ldots,k\}$ also violates* (6)*.*

We give the proof for the sake of completeness.

Proof. Proof. Let

$$\Gamma(K):=g(K)-\sum_{j\in K}p_j s_j$$

be the function that measures the violation of (6). Queyranne shows that for given s, if $K\subseteq N$ maximizes the violation $\Gamma(K)$ then $l\notin K$ if and only if $s_l\ge\sum_{j\in K}p_j$ (Queyranne, 1993, Lem. 5.2). Suppose K is a set that maximizes $\Gamma(K)$. Choose k such that $s_k<\sum_{j\in K}p_j$ and $s_{k+1}\ge\sum_{j\in K}p_j$. Then $j\in K$ for all $j\in\{1,\ldots,k\}$ and $j\notin K$ for all $j\in\{k+1,\ldots,n\}$, so $K=\{1,\ldots,k\}$. Therefore if there is a set that violates (6), i.e. $\Gamma(K)>0$, then there is an index k such that the set $\{1,\ldots,k\}$ maximizes that violation and thus also violates (6).

We next describe an algorithm that computes the point where a half-line from $x\in Q$ in direction $y\in\mathbb{R}^n$, $L=\{x+\lambda y\mid\lambda\in\mathbb{R}_{\ge 0}\}$ leaves the scheduling polytope Q. We also define $\ell(\lambda):=x+\lambda y$ and assume $y\neq 0$. The simple idea is this: The facet through which the half-line L leaves polytope Q is given by some subset K so that (6) ceases to be valid while moving along L in direction y. So we can enumerate all candidates for such K, and exploit the fact that, by Lemma 11, for given $\ell=\ell(\lambda)$ all candidate sets are among the nested sets $\{\sigma(1),\ldots\sigma(k)\}$, $k=1,\ldots,n$, where σ is the induced order of indices such that $\ell_{\sigma(1)}\le\cdots\le\ell_{\sigma(n)}$. This results in an efficient algorithm, as on L, there are in total at most $O(n^2)$ induced orders σ of the components of $\ell(\lambda)$.

Lemma 12. *The vectors $\ell(\lambda)$ on the line L have at most $O(n^2)$ different induced orders σ of their components.*

Proof. Proof. The relative order of $\ell(\lambda)_i$ and $\ell(\lambda)_j$ can change at most once, as L is a line.

By Lemma 12 we have no more than $O(n^2)$ induced orders on half-line L and it is not hard to see that they can all be computed in $O(n^3)$ total time. These $O(n^2)$ orders give rise to no more than $O(n^3)$ candidate sets K for the facet through which the half-line L leaves Q. We can compute the intersection of the facet induced by K with half-line L in time $O(n)$, and hence for any two candidates we can decide in time $O(n)$ which of the two intersections is closer to the given point $x \in Q$. By this line of argument we get an $O(n^4)$ time bound for computing a facet on which the half-line L leaves polytope Q. A slightly more clever bookkeeping, however, allows to obtain a better computation time.
The idea in improving the computation time is as follows. The relative order of $\ell(\lambda)_i$ and $\ell(\lambda)_j$ on L can change at most once for each pair of components i and j. For each such pair i, j with $y_i \neq y_j$ this order changes exactly when the components have equal value, i.e. at the point $\ell(\lambda(i,j))$, where

$$\lambda(i,j) = \frac{x_i - x_j}{y_j - y_i} \ .$$

For any pair i, j with $y_i = y_j$ the relative order of i and j is the same over the whole line L.
These points divide L into intervals I on which there is a single induced order σ of the components of the vectors $\ell \in I$. This not only bounds the number of induced orders by $O(n^2)$, it also bounds the total number of distinct nested sets $K = \{\sigma(1), \dots \sigma(k)\}$, $k = 1, \dots, n$, for all induced orders σ, by $O(n^2)$. This is because at $\ell(\lambda(i,j))$ only the relative order of i and j changes. If multiple pairs of components change relative orders in the same point $\ell(\lambda)$, we can treat these separately in the analysis to obtain the same result. This means that i and j are consecutive in the induced orders of the two intervals incident with the point $\ell(\lambda(i,j))$. Therefore all induced nested sets K for these two intervals are identical, except for the ones containing i and j.
Each of these induced nested sets K gives rise to one inequality (6) and for each such set K, for which $\sum_{j \in K} p_j y_j \neq 0$, the line L intersects the facet defined by (6) for K. Let us denote by $\delta(K)$ the parameter so that $\ell(\delta(K))$ is exactly this intersection point, and note that $\delta(K)$ can be easily computed if L and K are given. (Note that, if $\sum_{j \in K} p_j y_j = 0$ then the line L and the hyperplane induced by (6) for K are affinely dependent, and there is no intersection.) The values $\delta(K)$ can now be divided into two sets: those for which $q \geq \delta(K)$ for any $\ell(q) \in L \cap Q$ and those for which $q \leq \delta(K)$ for any $\ell(q) \in L \cap Q$. These provide lower bounds and upper bounds (on the line L) for the intersection of L and Q, respectively. The largest lower bound, denoted $\underline{\delta}$, and the smallest upper bound, denoted $\overline{\delta}$, exactly yield the intersection of L with Q.
To see why, note that for every $\ell(\lambda)$ with $\underline{\delta} \leq \lambda \leq \overline{\delta}$, (6) holds for all K that are induced nested sets of vector $\ell(\lambda)$. Therefore we have from Lemma 11 that $\ell(\lambda) \in Q$ and $\ell(\lambda) \in L$ by definition. Also, for any $\ell(\lambda)$ with $\lambda > \overline{\delta} = \delta(K)$ for some nested set K, (6) is violated for K and thus $\ell(\lambda) \notin Q$. Likewise for any $\ell(\lambda)$ with $\lambda < \underline{\delta}$, we have $\ell(\lambda) \notin Q$.
The idea is now to compute $\overline{\delta}$ and $\underline{\delta}$, together with the corresponding facet inducing nested sets K, by 'moving' along L in the order of sorted values $\lambda(i,j)$, and updating the nested sets K, and all other necessary parameters, incrementally.
The formal proof of the theorem is given along the lines of Algorithm 1, which gives the pseudocode for computing the intersection of a line L with the scheduling polytope Q.

Proof. Proof of Theorem 5. We annotate the pseudocode in Algorithm 1 and thereby derive the bound on the computation time. In line 3, the values $\underline{\delta}$ and $\overline{\delta}$, as well as $\mathcal{L}$ and $\mathcal{K}$ are initialized. $\mathcal{L}$ is the list of parameters $\lambda(i,j)$ on which the induced orders of $\ell(\lambda)$ change, and $\mathcal{K}$ is a container that contains all necessary information needed for the nested sets K and incremental updating in the course of the algorithm. The computation time of this initialization is $O(1)$.
In lines 4 to 9 the values $\lambda(i,j)$ are computed and added to $\mathcal{L}$. In line 10, $\mathcal{L}$ is sorted in ascending order. Since there are at most $n(n-1)/2$ of these values, the sorting can be done in time $O(n^2 \log n)$. In line 11, λ^0 is set to the smallest $\lambda(i,j)$ and in line 12 σ is set to be the order of $\ell(\lambda^0 - 1)$. This corresponds to the order of all $\ell(\lambda)$ with $\lambda < \lambda^0$. It can be computed in time $O(n \log n)$. Note that, if σ is stored as two arrays, one for $\sigma(\cdot)$ and one for $\sigma^{-1}(\cdot)$, calling either one requires time $O(1)$.
In lines 13 to 19 all nested subsets $K(j)$ that are induced by σ are stored in $\mathcal{K}$ together with the values $P(K(j))$, $F(K(j))$ and $Y(K(j))$. Note that $F(K(j)) - \lambda Y(K(j))$ is exactly equal to $\Gamma(K(j))$ for the point $\ell(\lambda)$. Therefore $\Gamma(K(j)) = 0$ for $\ell(\delta(K(j)))$. Computing the values $P(K(j))$, $F(K(j))$

and $Y(K(j))$ is done incrementally in time O(1). Since there are at most O(n) nested subsets $K(j)$, computing all of them can be done in time O(n).
In line 21 $\delta(K)$ is computed such that $\Gamma(K) = 0$ for $\ell(\delta(K))$ and the if clause on line 22 determines if (6) for K is satisfied by points $\ell(\lambda)$ for $\lambda > \delta(K)$, or by points $\ell(\lambda)$ for $\lambda < \delta(K)$. In the former case, $\delta(K)$ is an upper bound. In the latter case, $\delta(K)$ is a lower bound, which is then updated accordingly in lines 23 or 25. All steps can be performed in time O(1), there are O(1) computations per subset K and there are O(n) subsets, therefore all computations can be done in time O(n).
In lines 28 to 50 for each $\lambda(i, j)$ in ascending order we first determine how the order will change, i.e., whether i was before j or the other way around. Assume the former case, the latter case is symmetric. Then K is the σ^{-i}-th induced subset of σ, i.e. the subset containing i but not j. K' is computed as the new induced subset and $P(K')$, $F(K')$ and $Y(K')$ as the corresponding values. These replace $(K, P(K), F(K), Y(K))$ in $\mathcal{K}$, while i and j are switched in σ. Then the value $\delta(K')$ is computed and it is again determined if the upper or the lower bound has to be updated and this is done accordingly. Again each step can be performed in time O(1) and there are O(1) computations per iteration. There are O(n^2) iterations, therefore all computation can be done in time O(n^2).
If the returned values satisfy $\overline{\delta} < \underline{\delta}$, then the intersection of Q and L is empty. Otherwise the interval between $\ell(\underline{\delta})$ and $\ell(\overline{\delta})$ is the intersection of Q and L.
The computation time of the algorithm is dominated by the sorting of the O(n^2) values $\lambda(i, j)$ in line 10. So the total computation time of the algorithm is O($n^2 \log n$). We find the facets at which the line L and the scheduling polytope Q intersect by repeating lines 13 to 19 for $\overline{\delta}$ and $\underline{\delta}$.

Algorithm 1: Line Intersection Algorithm

Input : processing time vector $p \in \mathbb{R}^n_{>0}$, vectors $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$, $y \neq 0$.
Output: values $\underline{\delta}$ and $\overline{\delta}$.
$\underline{\delta} := -\infty$, $\overline{\delta} := \infty$, $\mathcal{L} := [\,]$, $\mathcal{K} := [\,]$.
for $i = 1$ **to** n **do**
  for $j = 1$ **to** $i - 1$ **and** $y_j \neq y_i$ **do**
    $\lambda(i,j) := (x_i - x_j)/(y_j - y_i)$
    $\mathcal{L} := \mathcal{L} + [((i,j), \lambda(i,j))]$
  end for
end for
Sort $\mathcal{L}$ increasing in $\lambda(i,j)$
$\lambda^0 := \lambda(i,j)$ of first element of $\mathcal{L}$
$\sigma :=$ order of $\ell(\lambda^0 - 1)$
for $j = 1$ **to** n **do**
  $K(j) := \{\sigma(1), \ldots, \sigma(j)\}$
  $P(K(j)) := P(K(j-1)) + p_j$
  $F(K(j)) := F(K(j-1)) + P(K(J-1))p_j - p_j x_j$
  $Y(K(j)) := Y(K(j-1)) + p_j y_j$
  $\mathcal{K} := \mathcal{K} + [(K(j), P(K(j)), F(K(j)), Y(K(j)))]$
end for
for $K \in \mathcal{K}$ **and** $Y(K) \neq 0$ **do**
  $\delta(K) := \frac{F(K)}{Y(K)}$
  if $F(K) - (\delta(K) + 1)Y(K) > 0$ **then**
    $\overline{\delta} := \min\{\overline{\delta}, \delta(K)\}$
  else
    $\underline{\delta} := \max\{\underline{\delta}, \delta(K)\}$
  end if
end for
for $(\lambda(i,j), (i,j)) \in \mathcal{L}$ **do**
  if $\sigma^{-1}(i) < \sigma^{-1}(j)$: **then**
    $K := \sigma^{-1}(i)$-th element in $\mathcal{K}$
    $K' := K \setminus \{i\} \cup \{j\}$
    $P(K') := P(K) - p_i + p_j$
    $F(K') := F(K) - (\frac{1}{2}P(K) - \frac{1}{2}p_i^2 - p_i x_i) + (\frac{1}{2}P(K') - \frac{1}{2}p_j^2 - p_j x_j)$
    $Y(K') := Y(K) - p_i y_i + p_j y_j$
  else
    $K := \sigma^{-1}(j)$-th element in $\mathcal{K}$
    $K' := K \setminus \{j\} \cup \{i\}$
    $P(K') := P(K) - p_j + p_i$
    $F(K') := F(K) - (\frac{1}{2}P(K) - \frac{1}{2}p_j^2 - p_j x_j) + (\frac{1}{2}P(K') - \frac{1}{2}p_i^2 - p_i x_i)$
    $Y(K') := Y(K) - p_j y_j + p_i y_i$
  end if
  Replace $(K, P(K), F(K), Y(K))$ in $\mathcal{K}$ by $(K', P(K'), F(K'), Y(K'))$
  Switch i and j in the order σ
  $\delta(K') := \frac{F(K')}{Y(K')}$
  if $F(K') - (\delta(K') + 1)Y(K') > 0$ **then**
    $\overline{\delta} := \min\{\overline{\delta}, \delta(K')\}$
  else
    $\underline{\delta} := \max\{\underline{\delta}, \delta(K')\}$
  end if
end for
return $\underline{\delta}, \overline{\delta}$

Bibliography

Alaei, Saeed, Hu Fu, Nima Haghpanah, Jason Hartline, Azarakhsh Malekian. 2012. Bayesian optimal auctions via multi- to single-agent reduction. *Proc. 13th ACM Conference on Electronic Commerce (EC 2012)*. ACM, 17.

Cai, Y., C. Daskalakis, S.M. Weinberg. 2012. Optimal multi-dimensional mechanism design: Reducing revenue to welfare maximization. *Proc. 53rd Annual Symposium on Foundations of Computer Science (FOCS 2012)*. IEEE Computer Society, 130–139.

Chawla, S., B. Sivan. 2014. Bayesian algorithmic mechanism design. *ACM SIGecom Exchanges* **13**(1) 5–49.

Conitzer, V., T. Sandholm. 2002. Complexity of mechanism design. A. Darwiche, N. Friedman, eds., *Uncertainty in Artificial Intelligence (UAI 2002)*. Morgan Kaufmann, 103–110.

Daskalakis, C., M. Weinberg. 2012. Symmetries and optimal multi-dimensional mechanism desig. *Proc. 13th ACM Conference on Electronic Commerce*. ACM, 370–387.

Duives, J., B. Heydenreich, D. Mishra, R. Müller, M. Uetz. 2015. On optimal mechanism design for a sequencing problem. *Journal of Scheduling* **18** 45–59.

Dyer, M.E., L.A. Wolsey. 1990. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics* **26**(2-3) 255–270.

Edmonds, J. 1971. Matroids and the greedy algorithm. *Mathematical Programming* **1**(1) 127–136.

Fishburn, P.C. 1992. Induced binary probabilities and the linear ordering polytope: a status report. *Mathematical Social Sciences* **23**(1) 67–80.

Fonlupt, J., A. Skoda. 2009. Strongly polynomial algorithm for the intersection of a line with a polymatroid. W. Cook, L. Lovász, J. Vygen, eds., *Research Trends in Combinatorial Optimization*. Springer, 69–85.

Gershkov, A., J.K. Goeree, A. Kushnir, B. Moldovanu, X. Shi. 2013. On the equivalence of bayesian and dominant strategy implementation. *Econometrica* **81**(1) 197–220.

Grötschel, M., L. Lovász, A. Schrijver. 1988. *Geometric algorithms and combinatorial optimization*. Algorithms and combinatorics, Springer.

Hart, S., N. Nisan. 2014. How good are simple mechanisms for selling multiple goods? Tech. Rep. DP-666, The Hebrew University of Jerusalem, Center for Rationality.

Hart, S., P. Reny. 2015. Maximal revenue with multiple goods: Nonmonotonicity and other observations. *Theoretical Economics* **10**(3) 893–922.

Hartline, J.D., A. Karlin. 2007. Profit maximization in mechanism design. N. Nisan, T. Roughgarden, É. Tardos, V. Vazirani, eds., *Algorithmic Game Theory*, chap. 13. Cambridge University Press, 331–362.

Heydenreich, B., D. Mishra, R. Müller, M. Uetz. 2008. Optimal mechanisms for single machine scheduling. C. Papadimitriou, S. Zhang, eds., *Internet and Network Economics (WINE 2008)*, *Lecture Notes in Computer Science*, vol. 5385. Springer, 414–425.

Hoeksma, R., B. Manthey, M. Uetz. 2014. Decomposition algorithm for the single machine scheduling polytope. P. Fouilhoux, L.E.N. Gouveia, A.R. Mahjoub, V.T. Paschos, eds., *Combinatorial Optimization (ISCO 2014)*, *Lecture Notes in Computer Science*, vol. 8596. 280–291.

Hoeksma, R., M. Uetz. 2013. Two dimensional optimal mechanism design for a sequencing problem. M. Goemans, J. Correa, eds., *Integer Programming and Combinatorial Optimization (IPCO 2013)*, *Lecture Notes in Computer Science*, vol. 7801. Springer, 242–253.

Kenis, P. 2006. Waiting lists in Dutch health care: an analysis from an organization theoretical perspective. *Journal of Health Organization and Management* **20**(4) 294–308.

Manelli, A. M., D. R. Vincent. 2010. Bayesian and dominant-strategy implementation in the independent private-values model. *Econometrica* **78**(6) 1905–1938.

Myerson, R. B. 1981. Optimal auction design. *Mathematics of Operations Research* **6**(1) 58–73.

Nisan, N. 2007. Introduction to mechanism design (for computer scientists). N. Nisan, T. Roughgarden, É. Tardos, V. Vazirani, eds., *Algorithmic Game Theory*, chap. 9. Cambridge University Press, 209–242.

Queyranne, M. 1993. Structure of a simple scheduling polyhedron. *Math. Programming* **58**(1-3) 263–285.

Queyranne, M., A. S. Schulz. 1994. Polyhedral approaches to machine scheduling. Technical Report 408/1994, TU Berlin.

Rothkopf, M. H. 1966. Scheduling with random service times. *Management Science* **12** 703–713.

Sandholm, T. 2003. Automated mechanism design: A new application area for search algorithms. F. Rossi, ed., *Principles and Practice of Constraint Programming (CP 2003)*, *Lecture Notes in Computer Science*, vol. 2833. Springer, 19–36.

Smith, W. E. 1956. Various optimizers for single-stage production. *Naval Research Logistics Quarterly* **3**(1-2) 59–66.

Vohra, R. V. 2011. *Mechanism Design - A Linear Programming Approach*. Econometric Society Monographs, Cambridge University Press.

Vohra, R. V. 2012. Optimization and mechanism design. *Mathematical Programming* **134**(1) 283–303.

Yasutake, S., K. Hatano, S. Kijima, E. Takimoto, M. Takeda. 2011. Online linear optimization over permutations. T. Asano, S.-I. Nakano, Y. Okamoto, O. Watanabe, eds., *Algorithms and Computation (ISAAC 2011)*, *Lecture Notes in Computer Science*, vol. 7074. Springer, 534–543.