Decentralized Sequencing of Jobs on a Single Machine

Erwin Pesch*,†, Mikhail Y. Kovalyov‡, Dominik Kress* and Sebastian Meiswinkel*

*University of Siegen, Management Information Science, Kohlbettstr. 15, 57068 Siegen, Germany,
Email: {erwin.pesch, dominik.kress, sebastian.meiswinkel}@uni-siegen.de

†Center of Advanced Studies in Management, HHL Leipzig, Jahnallee 59, 04109 Leipzig, Germany

‡United Institute of Informatics Problems, National Academy of Sciences of Belarus, 220012 Minsk, Belarus,
Email: kovalyov_my@newman.bas-net.by

***Abstract*—There is a finite number of non-cooperating clients competing for execution of their jobs by a single service provider in order to minimize job completion time costs. The clients can move their jobs to complete earlier in a given sequence. However, they have to compensate the cost increase to the other clients whose jobs are completed later due to this move. All clients are assumed to be fully risk averse. A game mechanism is suggested, such that no client has an incentive to claim false cost and a social criterion, i.e. the minimization of the total cost of all clients, is addressed. A polynomial time algorithm that finds a game equilibrium is suggested and embedded into the game mechanism. Computational tests analyze the performance and practical suitability of the resulting mechanism. We outline potential directions for future research in a similar setting in a parallel service environment.**

I. Introduction

Many service providers apply a *centralized decision* to determine the timing of their clients' processing. A typical example involves logistics operators at seaports or rail-road terminals. A drawback of this setting is that the clients are not always satisfied with the centralized decision. They oftentimes would like to have an influence on their positions in the processing sequence. In this context, this paper analyzes a *game decision mechanism* that has been introduced in [1]. This mechanism is such that the clients are involved in the decision process but are forced to be truthful if they do not accept a risky decision. Furthermore, it addresses a *social criterion*.

A. Notation and Problem Description

We study a problem in which n clients, each having a single job, compete for early execution of their jobs in the processing sequence of a single service provider that processes jobs one after another with no idle time between them. We denote the set of clients by $N = \{1, \ldots, n\}$ and refer to job j as the "job of client" j.

The clients are assumed to be non-cooperative. They therefore cannot form coalitions to exchange information and generate a group decision. Furthermore, we assume that the clients are rational and *fully risk averse*. That is, they do not lie if this may induce a decrease of their utility function or if lying does not increase their utility.

All jobs are ready for processing at time zero. A processing time p_j and a non-decreasing cost function $f_j(t)$ are associated with job j, $j = 1, \ldots, n$. The values p_j, $j = 1, \ldots, n$, are truly claimed by the service provider because his revenue from processing any job is fixed. The cost function $f_j(t)$, however, is claimed by client j and can differ from the true cost function, $f_j^{true}(t)$, $j = 1, \ldots, n$. Because of business confidentiality constraints, the values p_j, $j = 1, \ldots, n$, are not revealed to the clients, and the cost functions $f_j(t)$ and $f_j^{true}(t)$ of client j are not revealed to the other clients.

In [1], a game mechanism for the above setting is suggested. This mechanism determines an initial job processing sequence as well as rules of moving jobs to earlier positions within this sequence. If a job is moved, the corresponding client is obliged to compensate the cost increase to the other clients whose jobs are completed later due to this move. After a certain number of moving operations, a final sequence is obtained and the jobs are processed in this sequence.

We denote the completion time of job j in a given job sequence by C_j. It corresponds to the sum of the processing times of all jobs preceding and including job j. Client j aims at maximizing the *utility function* $F_j := V_j^+ - V_j^- - f_j^{true}(C_j)$, where V_j^+ is the total compensation paid to him and V_j^- is the total compensation paid by him, $j = 1, \ldots, n$. Note that $\sum_{j=1}^{n}(V_j^+ - V_j^-) = 0$.

We consider minimizing the total claimed cost of all clients, $\sum_{j=1}^{n} f_j(C_j)$, as a social criterion that the service provider would like to address. The problem of finding a job sequence that minimizes the total cost is denoted by $1||\sum f_j(C_j)$ in the scheduling literature; see [2]. For an introduction to scheduling problems in general, see [3].

B. Applications and Related Literature

Consider a cargo carrier company which possesses a sea liner to perform voyages from a single base port to several destination ports. Each voyage is direct in the sense that its route is from the base port to a destination port and back to the base port. At the beginning of a financial year, the carrier negotiates n long-term service contracts with the shipping companies. Each contract specifies the delivery of a given

cargo from the base port to one of the destination ports. The return trips are filled with cargo of the spot market. Due to business constraints, cargo of different long-term contracts cannot be assigned to the same voyage. For the shipping company, each long-term contract j is associated with the cargo target delivery date D_j and an extra profit w_j, which the shipping company earns if and only if this cargo is delivered by D_j. For the carrier, a long-term contract is associated with the direct trip duration a_j, the backward trip duration b_j, and a fixed revenue. Revenues from the spot market contracts are not considered here. During the negotiation process, the carrier would like to work out a timetable for the n voyages which is satisfactory to the interested shipping companies, and then to specify it in the corresponding contracts.

In terms of the problem we study, the long-term contracts are the jobs, the total trip durations a_j+b_j are the job processing times p_j, and $f_j^{true}(t) = w_jU_j$ is the true cost function associated with the long-term contract j, where $U_j = 0$ if $t - b_j \leq D_j$, or equivalently, $t \leq d_j := D_j + b_j$, and $U_j = 1$ if $t > d_j$, $j = 1, \ldots, n$. Here, if job j completes at time t, then the corresponding cargo arrives at its port at time $t - b_j$. The problem of minimizing the sum of functions w_jU_j is denoted by $1||\sum w_jU_j$, values d_j are called *due dates* and values w_j are called *weights* in the scheduling literature; see [2].

Another example is the problem of determining an execution sequence for n computer tasks accepted by a computing service provider between two adjacent time points of making a decision of task acceptance or rejection and consequent execution of accepted tasks. Each accepted task j is associated with its execution time p_j and the money value w_j lost by the task owner per unit of time elapsed since the acceptance decision is made (time zero) and before its execution completes (time C_j). The owner of task j would like to minimize the total loss w_jC_j. The corresponding scheduling problem to minimize $\sum w_jC_j$ is denoted by $1||\sum w_jC_j$; see [2].

Lately, there has been a growing interest in decentralized supply chain management decisions and their comparison with centralized decisions; see, for example, [4]–[9]. Studies of decentralized decisions in scheduling deal with application areas as, for example, computer grids and clouds [10], parallel and distributed computer systems [11], crane scheduling [12], semiconductor manufacturing [13], resource constrained multi-project scheduling [14], mobile robots scheduling [15], and supply chain coordination [16]. Game-theoretic models are often employed to handle decentralized scheduling problems. We refer to [1] for the relevant references.

C. Contribution and Structure of this Paper

In [1], the authors focus on theoretical insights. They do not provide a computational analysis of the game mechanism. The main contribution of this paper is to close this gap by performing a corresponding computational study. The computational results demonstrate that the game mechanism enables a service provider to provide high quality solutions that feature an appropriate level of communication between the service provider and the clients. We will furthermore outline potential directions for future research in a similar setting in a parallel service environment.

The remainder of this paper is structured as follows. In Section II, we will present the game mechanism described in [1] along with some important insights. The computational study is then presented in Section III. Future research directions are given in Section IV.

II. Game Decision Mechanism

In [1], we propose the following game mechanism for the service provider to find a job sequence which is satisfactory to all clients.

At the beginning, the clients claim cost functions $f_j(t)$, $j = 1, \ldots, n$. The functions are assumed to be represented such that a constant number of elementary arithmetic operations is needed to calculate any value $f_j(t)$ for $j \in \{1, \ldots, n\}$ and $0 \leq t \leq \sum_{i=1}^{n} p_i$.

The mechanism is a decision process that generates a final job sequence. First, an initial job sequence is generated. Any approach can be used here. For example, if the clients would like to have equal chances to take any position in the initial sequence, it can be randomly generated. If the clients agree that the mechanism applies any rule to generate the initial sequence, then we suggest that it is the best sequence with respect to minimizing the total claimed cost $\sum_{j=1}^{n} f_j(C_j)$, which the mechanism can find within a given time limit. The rules of developing this sequence can be known to the clients or not.

Then, if the initial sequence was developed aimed at minimizing the total cost, the mechanism updates the initial sequence by swapping the jobs in the first and the second positions. This is done to prevent any client from claiming a false cost function in order to take the first position such that it cannot be taken by any other job because of a large compensation payment.

Denote the updated sequence by $S^{old} = (i_1, \ldots, i_n)$. S^{old} is the input sequence for the first iteration of the decision process. Each client i_j, whose job is not in the first position, receives the completion time of his job in this sequence and a set of possible completion times obtained by placing his job in every earlier position assuming that the relative sequence of the other jobs remains unchanged. He also receives a set $E(i_j)$ of *eligible local strategies*. For a client i_j, whose job is not in the last position, this set includes every strategy (i_j, s) of moving his job to an earlier position s, $1 \leq s \leq j-1$, in S^{old}, such that his *claimed savings* that emerge from applying this strategy are positive. The definition of claimed savings is given below.

Denote by S^{new} the sequence obtained from S^{old} by applying a job moving strategy (i_j, s), $1 \leq s \leq j-1$. The claimed savings $D(i_j, s)$ of client i_j associated with this strategy are calculated as follows:

$$D(i_j, s) = A - B, \tag{1}$$

where

$$A = f^{old}(i_j) - f^{new}(i_j)$$

is the reduction of the claimed cost of client i_j, and

$$B = \sum_{k=s}^{j-1} \left(f^{new}(i_k) - f^{old}(i_k) \right)$$

is the *compensation* of client i_j to the other clients. $f^{old}(i_k)$ and $f^{new}(i_k)$, $k = s, s+1, \ldots, j$, are the claimed costs of client i_k, calculated using completion time of job i_k in the sequences S^{old} and S^{new}, respectively.

If every job moving strategy results in non-positive claimed savings for client i_j, then "no move" ($E(i_j) = \{(i_j, j)\}$) is the only eligible strategy. For the client, whose job is last, the set $E(i_n)$ consists of the job moving strategies with positive claimed savings and the "no move" strategy (i_n, n) because staying last does not affect actions of other clients.

Each client i_j submits one eligible local strategy of the set $E(i_j)$, $2 \leq j \leq n$. The service provider selects and applies one of them. Again, any approach to the selection can be used here, for example, random selection. If the clients agree that the mechanism applies any selection rule, then we suggest that it selects the strategy that minimizes the total claimed cost $\sum_{j=1}^{n} f_j(C_j)$. The resulting sequence serves as the input sequence S^{old} in the next iteration of the decision process. When making a choice, the client can rank his eligible local strategies. For example, he may consider maximizing savings as the choice criterion. Alternatively, moving closer to the beginning of the sequence can be the choice criterion. Note that these criteria can be contradictory.

The decision process is repeated until a final job sequence is obtained, for which no set $E(i_j)$, $j = 2, \ldots, n$, contains an eligible job moving strategy different from "no move". The jobs are processed by the service provider in the order determined by the final job sequence. All compensation payments are realized.

We call the described process as a *sequence updating game with compensations and jobs competing for earlier positions*. It is summarized as follows:

Step 1 Each client $j \in \{1, \ldots, n\}$ claims a cost function $f_j(t)$.

Step 2 Calculate an initial job sequence S with a given algorithm. Depending on the algorithm used, potentially update S by interchanging the jobs in the first and second position of the job sequence.

Step 3 Determine the set $E(i_j)$ of eligible local strategies for each client i_j. If $E(i_j) = \{$"no move"$\}$ for all clients i_j, then go to Step 4. Else, each client i_j with $E(i_j) \neq \{$"no move"$\}$ submits exactly one local strategy. Apply one of the strategies, which is selected based on a given algorithm, to update schedule S. Update the compensation payments of all players. Go to Step 3.

Step 4 Apply sequence S. Realize the compensation payments.

In this game, clients are *players*. We define an *Equilibrium (EQ)* of this game as a job sequence such that no client can obtain positive claimed savings by applying an eligible job moving strategy.

A. Truthfulness of Clients

In the described game the clients can claim false cost functions if there is no risk that this action will decrease their utility. Let us show that no client has an incentive to do this.

Consider a client j who lies about his cost function. Recall that all the cost functions are claimed before the sequence updating procedure starts. In any iteration of this procedure, including the first iteration, there is a chance that job j is not in the first position. Assume that job j is not in the first position and a certain strategy is applied. There are three cases to consider: 1) job j is moved to an earlier position, 2) job j is moved to a later position, 3) the position of job j does not change.

In case 3), the utility of client j does not change irrespectively of the claimed cost function.

In case 1), denote by S^{false} the claimed savings of client j. Denote by S^{true} his claimed savings calculated for the true cost function and the same earlier position. Note that $S^{false} > 0$ because the applied move is an eligible strategy. If $S^{false} \leq S^{true}$, then $S^{true} > 0$. This implies that the same move would be eligible for the true cost function of client j and, moreover, his utility would be the same or larger than that for the false cost function.

Assume $S^{false} > S^{true}$. Since the cost functions and the job processing times of other clients are unknown and arbitrary for client j, there is a risk that, in the case of the false cost function, job j overtakes jobs whose cost increase client j is not able to compensate from his true cost reduction, i.e., $S^{true} < 0$. In this case, lying can decrease the utility of client j. For example, let job j be sequenced immediately after job i, $p_i = p_j$, both jobs have the same due date which is equal to the completion time of job i, the true and claimed weights of job j be equal to w_j and $w_j + \delta$, respectively, and the claimed weight of job i be equal to $w_j + \delta/2$. The cost functions are $w_i U_i$ and $w_j U_j$. Assume that job j is moved to stand immediately before job i. Then the claimed savings of client j are $S^{false} = \delta/2$, while his true savings are $-\delta/2$.

In case 2), client j is paid by another client whose job is moved to stand before job j. Denote by I^{false} and I^{true} the cost increase of client j calculated with respect to his claimed and true cost functions, respectively. The compensation paid to client j in case 2) is equal to I^{false}, and the decrease of his utility function is equal to I^{true}. If $I^{false} \leq I^{true}$, then lying may not be profitable. Furthermore, there is no guarantee that case 2) in which $I^{false} > I^{true}$ will always happen, because a large cost increase of client j can make job moving strategies of other clients ineligible. Hence, case 1) can happen and the client can loose. Consider the example from the previous paragraph. Now assume that job j is sequenced immediately before job i and that job i is moved to stand immediately before job j. In this situation, the cost increase of client j with respect to his claimed cost function is equal to $I^{false} = w_j + \delta$ and the cost increase with respect to his true

cost function is equal to $I^{true} = w_j$. Job i cannot overtake job j in the case of its false cost function because savings of client i are equal to $-\delta/2$ in this case.

Thus, we have shown that no client has an incentive to claim a false cost function, unless he is risky. We assumed that the clients do not take a risky decision. Therefore, in the suggested game they should claim true cost functions.

B. Implementing Step 2 of the Game Mechanism

An interesting question when analyzing the sequence updating game is as follows: Does it pay off to carefully design an algorithm to be applied in Step 2, with respect to the number of iterations of Step 3 that the game mechanism needs to terminate? The importance of this question stems from the fact that in potential applications one typically wants to avoid being faced with communication intensive processes. If $1||\sum f_j(C_j)$ is NP-hard, it seems to be a natural choice to develop a polynomial time heuristic algorithm that terminates with an EQ. In this case, one may hypothesize that this results in only a few iterations of Step 3.

For the remainder of this paper, we will consider cost functions of the form w_jU_j, $j = 1, \ldots, n$ (cf. already Section I for the related notation and an application). It is well known that $1||\sum w_jU_j$ is NP-hard [17]. Now, consider the following Algorithm, which we refer to as *GreedyWeight*:

Step 1 Renumber jobs such that $w_1 \geq \cdots \geq w_n$, breaking ties arbitrarily. Initialize an empty sequence of early and on-time jobs S^{early} and am empty sequence of late jobs S^{late}. Set $k := 1$.

Step 2 Add job k to the sequence S^{early}. Re-arrange jobs of this sequence in their EDD order, breaking ties arbitrarily. If at least one job in the sequence S^{early} does not complete by its due date, remove job k from S^{early} and add it to an arbitrary position of the sequence S^{late}. If $k = n$, then return $S^0 := (S^{early}, S^{late})$ and stop. If $k \leq n-1$, then set $k := k+1$ and repeat Step 2.

GreedyWeight constructs a job sequence S^0. Note that the *Earliest Due Date (EDD)* order of jobs is such that a job with smaller due date appears earlier than a job with larger due date.

GreedyWeight can be implemented to run in $O(n^2)$ time as follows. Let $S^{early} = (i_1, \ldots, i_r)$ be the sequence of early and on-time jobs arranged in their EDD order at the beginning of iteration k of Step 2. With each job of this sequence, store its completion time C_{i_j}, $j = 1, \ldots, r$. When job k is added, the sequence S^{early} can be updated in $O(\log n)$ time to keep the EDD order by a bisection search over the range $1, \ldots, r$ of job positions. In each iteration of the bisection search, d_k is compared with d_{i_t} for a trial value $t \in \{1, \ldots, r\}$. Let job k be inserted between jobs i_{h-1} and i_h. The completion times of jobs $i_1, \ldots, i_{h-1}$ remain unchanged, the completion time of job k is equal to $C_k = C_{i_{h-1}} + p_k$ and the completion times of jobs $i_h, \ldots, i_r$ increase by p_k. These computations require $O(n)$ time. Having job completion times, the feasibility of the new sequence with respect to the due dates can be verified in $O(n)$ time. Thus, each iteration of Step 2 requires $O(n)$ time, and the run time of algorithm GreedyWeight is $O(n^2)$.

Proposition 1. *S^0 is an EQ for cost functions $f_j(C_j) = w_jU_j$, $j = 1, \ldots, n$.*

Proof: Consider the sequence S^0 returned by GreedyWeight and denote the set of jobs in the sequence S^{early} in iteration k of Step 2 by $X(k)$. Furthermore, denote the social value of a given job sequence S by $F(S)$. Observe that, by definition of GreedyWeight and the fact that the EDD sequence minimizes maximum job tardiness, if job k is late in S^0, then at least one job of the set $X(k)$, which comprises this job and all early and on-time jobs considered before job k in Step 2, will be late in any sequence. Assume that S^0 is not an EQ. Then there exists a job k', such that $F(S') < F(S^0)$ for the sequence S' obtained from S^0 by moving job k' to an earlier position. Consider the jobs of the set $X(k)$ in the sequence S'. We have shown that at least one job of this set is late in any sequence, including S'. By definition of this set, the weight of this job is at least w_k. Since the cost reduction of job k' emerged from its moving to an earlier position is at most w_k, $F(S') \geq F(S^0)$, which is a contradiction. Hence, S^0 is an EQ. ∎

Another interesting property of GreedyWeight is proven in [1]. Denote by F^{Opt} the objective function value of a social optimum Opt. An algorithm for the problem $1||\sum f_j(C_j)$ is a Δ-*approximation* algorithm if it computes a solution with value $F^0 \leq \Delta F^{Opt}$ for any problem instance. The bound Δ is called *tight* if $F^0 = \Delta F^{Opt}$ for some instance of the problem.

Proposition 2. *Algorithm GreedyWeight is a $(n-1)$-approximation algorithm for the problem $1||\sum w_jU_j$ and this bound is tight.*

Note that there exist polynomial time approximation algorithms for the problem $1||\sum w_jU_j$ with approximation ratio $\Delta < n-1$, in particular an FPTAS presented in [18], in which $\Delta = 1 + \varepsilon$ for any given $\varepsilon > 0$. However, it is not known if any of these algorithms produces an EQ.

Moreover, there exist other algorithms that solve $1||\max\{w_jU_j\}$ in $O(n^2)$ time, e.g. Lawler's algorithm [19]. Lawler's algorithm repeatedly assigns a job with minimum cost $f_j(P_k)$ to position k, where P_k is the total processing time of unscheduled jobs, $k = n, n-1, \ldots, 1$. Ties are broken arbitrarily.

The following example shows that Lawler's algorithm may produce a solution which is not an EQ and which cannot be transformed to an EQ by moving a single job to an earlier position. In the example, there are three jobs with parameters $p_1 = 3$, $p_2 = 2$, $p_3 = 1$, $d_1 = 2$, $d_2 = d_3 = 3$ and $w_1 = w_2 = w_3 = 1$. For this example, Lawler's algorithm may output sequence $(1, 2, 3)$, which is not an EQ because the savings from moving job 2 or job 3 to the first position are equal to 1 (the cost of the moved job reduces by 1 and the costs of the other jobs do not change). None of the sequences

(3,1,2), (1,3,2) and (2,1,3) that are obtained from (1,2,3) by moving one job to an earlier position is an EQ as well. The EQ sequences are (2,3,1) and (3,2,1).

III. Computational Study

With regard to our research question, we conducted a series of computational experiments on randomly generated instances to analyze the quality of GreedyWeight with respect to the social criterion $\sum_{j=1}^{n} w_j U_j$ when being used in Step 2 of the sequence updating game. We implemented all algorithms in C++ and performed the computational experiments on a PC with 16 GB of memory and an Intel® Core™ i7 CPU, running at a speed of 3.4 GHz. The operating system was Windows 8.1, 64 bit.

In order to generate benchmark solutions, we implemented three versions of the sequence updating game. They differ in the way they generate the initial job sequence (Step 2). We implemented a random generation procedure, algorithm GreedyWeight, and a complete enumeration approach that determines optimal sequences of the underlying scheduling problem $1||\sum w_j U_j$. We refer to the resulting versions of the sequence updating game by $game^{rand}$, $game^{GW}$ and $game^{emum}$, respectively. With respect to Step 3 of the sequence updating game, we assume that the clients act in a greedy manner, i.e. they always pick a move with largest saving among their list of eligible moves. We additionally assume that the service provider applies a strategy with largest saving out of the set of strategies that have been submitted by the clients. The sequence updating game terminates if no player has an eligible move different from "no move", i.e. when an EQ is reached.

We generated two groups of test instances. The first group features small instances with $n \in [10, 30]$, for which we were able to generate optimal initial sequences with respect to the social criterion $\sum_{j=1}^{n} w_j U_j$ with the complete enumeration approach. The second group relates to large instances with $n \in [40, 400]$, that we could not solve to optimality. For both groups, processing times p_j, weights w_j, and due dates d_j, $j \in N$, were randomly drawn from uniform distributions over the intervals $[1, 100]$, $[100, 200]$, and $[p_j, P/m]$, respectively, where $P := \sum_{j=1}^{n} p_j$. If $p_j > P/m$, we set $d_j = p_j$.

For each n, we generated a total of ten test instances. For each of the resulting instance sets, we rate the performance of a specific variant of the sequence updating game by calculating the arithmetic mean of the corresponding solution qualities, which are defined as F^*/F', where F^* is the total scheduling cost of the solution determined by the specific variant of the game mechanism, and F' is the scheduling cost of the best solution obtained by any of the considered algorithms. Note that the algorithms that apply complete enumeration are solely considered in the first test set.

As mentioned above, we specifically want to analyze the number of iterations that the game mechanism needs to terminate, because the corresponding updating procedures include potentially time consuming communication processes between the clients and the operator. Hence, runtime comparisons are of minor interest, which is supported by the fact that all variants of the sequence updating game, except the ones that apply the complete enumeration approach, terminated in less than 0.6 seconds, even for the largest instances with $n = 400$, in our computational tests. For our comparisons, we use the arithmetic mean of the number of iterations that the game mechanism needs to terminate for each set of instances.

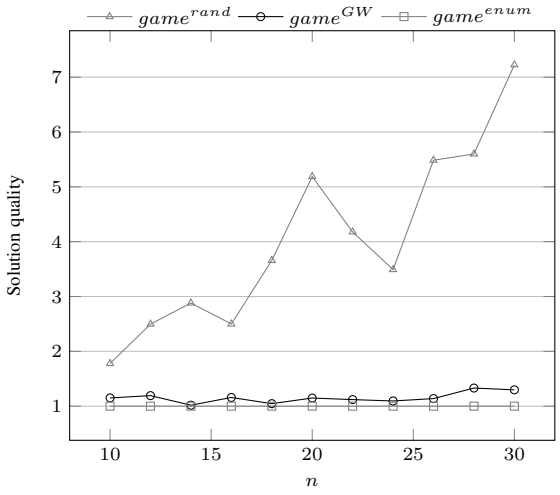

Fig. 1. Small instances - solution quality

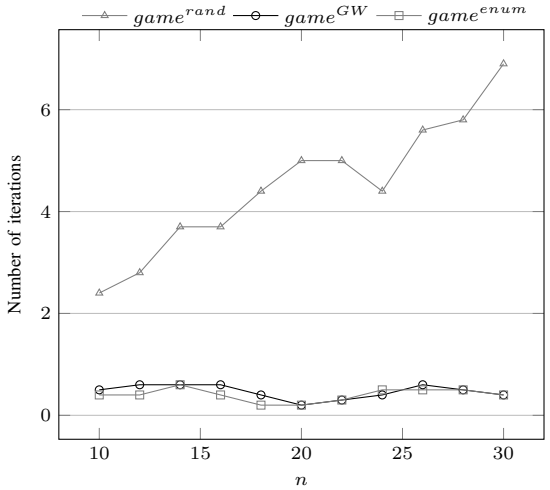

Fig. 2. Small instances - number of iterations of Step 3

We will first analyze the results for the small instances. In Fig. 1, we plot the algorithms' corresponding average qualities over the number of jobs.

As to be expected, integrating the complete enumeration approach results in the best overall solution quality. However, applying GreedyWeigtht to determine an initial sequence results in fairly good solution qualities. Moreover, in comparison to the random generation of initial solutions, the average solution quality degrades significantly slower when increasing the number of jobs.

Fig. 2 plots the average number of iterations of Step 3 over the number of jobs for the small instances.

The algorithms based on an initial sequence generated by

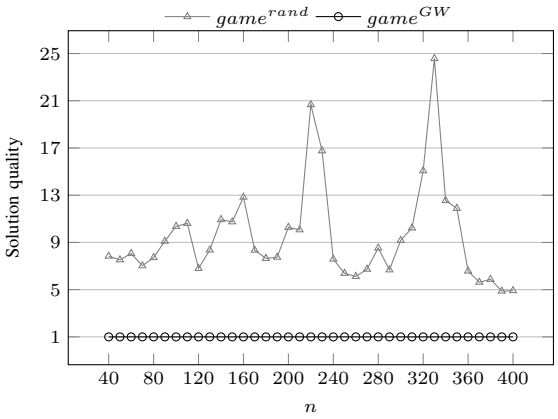

Fig. 3. Large instances - solution quality

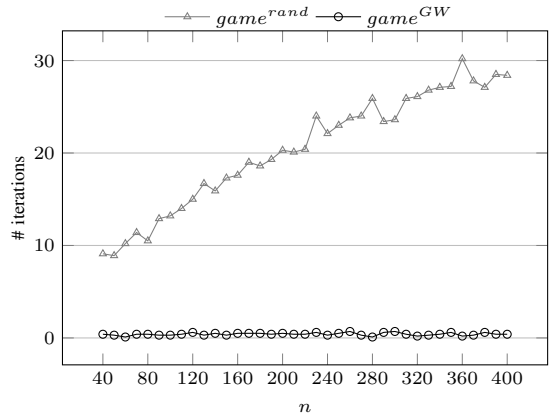

Fig. 4. Large instances - number of iterations of Step 3

GreedyWeight or by complete enumeration clearly outperform $game^{rand}$. We believe that the former is a result of Proposition 1, i.e. the fact that the solutions determined by GreedyWeight are EQs. These solutions are only slightly modified in Step 2 of the game mechanism, so that less iterations of Step 3 are needed in order to restore the EQ.

We will now turn our attention to the large test instances. Fig. 3 plots the corresponding results on the average solution quality.

Again, generating initial sequences by using GreedyWeight clearly pays off when compared to random generation of initial sequences.

Fig. 4 depicts the average number of iterations of Step 3 of the sequence updating game over the number of jobs for the large instances.

Again, applying GreedyWeight outperforms random generation of initial sequences. While, on average, $game^{GW}$ needs less than 1 iteration for all considered n, the number of iterations of Step 3 seems to increase almost linear in n for $game^{rand}$. This substantiates our believe that the game mechanism performs best when carefully generating an initial sequence that is an EQ.

IV. Summary and Future Research

In this paper, we have analyzed a game mechanism that has recently been introduced in [1] for sequencing jobs on a single machine. Applications arise in planning problems at logistics companies, as for example cargo carriers. Our main result is that this game mechanism results in high quality solutions when it is carefully designed with respect to generating initial job sequences. In this case, it is moreover well suited from a practical perspective because it results in an appropriate level of communication between operator and clients.

For future research it will especially be interesting to generalize the setting presented in this paper to the case of more than one machine, i.e. assuming that there is a finite number of non-cooperating clients competing for execution of their jobs not later than by their respective due dates in a parallel service environment. This may be motivated by planning operations of a railway container unloading terminal (see, for example, [20]). There are m parallel railway tracks and each track is served by an associated single hoist crane. Each crane can process at most one container train at a time. For a given planning period, the terminal financial manager negotiates n service contracts with the shipping agents. Each contract specifies the unloading of a train. For a shipping agent j, the contract is associated with the train unloading due date d_j^{true} and a cost w_j^{true}, which is paid if and only if the train misses the due date d_j^{true}. For the terminal owner, this contract is associated with the train unloading time p_j and a fixed revenue. During the negotiation process, the manager would like to work out a schedule for unloading n trains by m cranes which is satisfactory to the interested shipping agents, and then to specify it in the corresponding contracts.

Other generalizations and modifications of the sequencing environment can also be considered. For example, precedence constraints on the set of jobs are an interesting path for future research. Moreover, it is interesting to analyze clients that accept risky decisions. Finally, there is a wide variety of cost and objective functions that can be analyzed in analogy to the above case, where we considered cost functions of the form $w_j U_j$, $j = 1, \ldots, n$.

References

[1] M. Y. Kovalyov and E. Pesch, "A game mechanism for single machine sequencing with zero risk," *Omega*, vol. 44, pp. 104–110, 2014.

[2] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[3] J. Błażewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, *Handbook On Scheduling: From Theory to Applications*. Berlin: Springer, 2007.

[4] O. Kaya, "Outsourcing vs. in-house production: a comparison of supply chain contracts with effort dependent demand," *Omega*, vol. 39, no. 2, pp. 168–178, 2011.

[5] T. Hosoda and S. M. Disney, "A delayed demand supply chain: Incentives for upstream players," *Omega*, vol. 40, no. 4, pp. 478–487, 2012.

[6] Jonrinaldi and D. Zhang, "An integrated production and inventory model for a whole manufacturing supply chain involving reverse logistics with finite horizon period," *Omega*, vol. 41, no. 3, pp. 598–620, 2013.

[7] A. Varmaz, A. Varwig, and T. Poddig, "Centralized resource planning and yardstick competition," *Omega*, vol. 41, no. 1, pp. 112–118, 2013.

[8] Q. Qiang, K. Ke, T. Anderson, and J. Dong, "The closed-loop supply chain network with competition, distribution channel investment, and uncertainties," *Omega*, vol. 41, no. 2, pp. 186–194, 2013.

[9] J. Zhang, W.-Y. K. Chiang, and L. Liang, "Strategic pricing with reference effects in a competitive supply chain," *Omega*, vol. 44, pp. 126–135, 2014.

[10] Y. Huang, N. Bessis, P. Norrington, P. Kuonen, and B. Hirsbrunner, "Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 402–415, 2013.

[11] M. Tchiboukdjian, N. Gast, and D. Trystram, "Decentralized list scheduling," *Annals of Operations Research*, vol. 207, no. 1, pp. 237–259, 2013.

[12] O. Sharif and N. Huynh, "Yard crane scheduling at container terminals: A comparative study of centralized and decentralized approaches," *Maritime Economics & Logistics*, vol. 14, no. 2, pp. 139–161, 2012.

[13] S. Yao, Z. Jiang, N. Li, N. Geng, and X. Liu, "A decentralised multi-objective scheduling methodology for semiconductor manufacturing," *International Journal of Production Research*, vol. 49, no. 24, pp. 7227–7252, 2011.

[14] J. Homberger, "A (μ, λ)-coordination mechanism for agent-based multi-project scheduling," *OR Spectrum*, vol. 34, no. 1, pp. 107–132, 2012.

[15] S. Giordani, M. Lujak, and F. Martinelli, "A distributed multi-agent production planning and scheduling framework for mobile robots," *Computers & Industrial Engineering*, vol. 64, no. 1, pp. 19–30, 2013.

[16] X. Qi, J. F. Bard, and G. Yu, "Supply chain coordination with demand disruptions," *Omega*, vol. 32, no. 4, pp. 301–312, 2004.

[17] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*. New York: Plenum, 1972, pp. 85–103.

[18] G. V. Gens and E. V. Levner, "Fast approximation algorithm for job sequencing with deadlines," *Discrete Applied Mathematics*, vol. 3, no. 4, pp. 313–318, 1981.

[19] E. L. Lawler, "Optimal sequencing of a single machine subject to precedence constraints," *Management Science*, vol. 19, no. 5, pp. 544–546, 1973.

[20] D. Kress, S. Meiswinkel, and E. Pesch, "The partitioning min-max weighted matching problem," *European Journal of Operational Research*, vol. 247, no. 3, pp. 745–754, 2015.