Scheduling Output Jobs in regards of General Device Incompatibility Constraints is hard

Uwe Keller
firstname.lastname@mailserviceofgoogle.com

December 27, 2016

Abstract

We define the computational problem of output job scheduling and show that the optimizaton version of the problem is **NP**-hard and the the decision version of the problem is **NP**-complete. Consequently, our result strongly suggests that there is no polynomial time algorithm to solve the information scheduling problem, unless $\mathbf{P} = \mathbf{NP}$ [1] – which (since decades) is widely believed to be not true. Hence, one should focus on the design of algorithms that find non-optimal (but correct) solutions to the problem instances instead.

Further, our proof also holds for any restricted class of graphs for which Independent Set is **NP**-hard, thus identifying more restricted cases, where the output job scheduling problem already is **NP**-hard.

1 Problem Definition

Consider a number of information output tasks (say information messages to be shared at train stations to keep travellers informed about travel related events). Abstracting from the concrete information to be shared, each of these output tasks (also refered to *output jobs* in the following) has assigned a desired start time, an expected output duration and a target device to render the output job. Rendering of an output job might happen for instance graphically on a display or acoustically as an announcement on a loud speaker. During the output period of an output job, the device will render the content of the output job at least from the moment the output job output starts to the first moment the output ends or some other output job output starts on the same device.

Certain devices might be *incompatible* with each other: For instance, using loudspeakers to acoustically render (i.e. announce) different messages at the same moment in time might be not be desired if the loud speakers are too close to each other, since travelers might not be able to understand the announcement relevant to them because of a simultaneous closeby announcement. Futher, it might in general not be desirable for any two different messages announced on the same loudspeaker to overlap in time. To deal with such undesirable overlaps between output jobs, we consider as the only course of action to delay the start of an output jobs such that overlaps do not occur. This leads to the following computation problem:

The *output job scheduling problem* is about determining a correct and complete schedule to output output jobs under the presence of device incompatibility constraints such that the total sum of delays over all output jobs (in regards of the desired output time) is minimized. A schedule is complete, if all considered output jobs are scheduled. A schedule for output jobs is correct, if there are no two different output jobs in the schedule that target conflicting devices and are overlapping according the schedule.

Formally, we can define the output job scheduling problem as follows

Definition 1 (OUTPUT JOB SCHEDULING Optimization Problem). *Given a finite set* $C = \{c_1, \ldots, c_n\}$ *of output jobs, a finite set of device names* $D = \{d_1, \ldots, d_k\}$*, total functions*

- $start : C \to \mathbb{N}_0$ *for desired start times,*
- $duration : C \to \mathbb{N}_0$ *for the expected output duration,*
- $device : C \to D$ *for the target output device*

and an incompatibility relation $I \subseteq D \times D$*, find a total function* $\sigma : C \to \mathbb{N}_0$ *called* schedule *that satisfies all of the following (correctness) constraints:*

- $start(c) \leq \sigma(c)$ *for all* $c \in C$*, and*
- *for all* $c_1, c_2 \in C$ *whenever* $c_1 \neq c_2$ *and* $(device(c_1), device(c_2)) \in I$ *then* c_1 *and* c_2 *are non-overlapping in* σ*, i.e.* $\sigma(c_1)+duration(c_1) \leq \sigma(c_2)$ *or* $\sigma(c_2)+duration(c_2) \leq \sigma(c_1)$

and minimizes the objective funtion

$$delay(\sigma) = \sum_{c \in C} (\sigma(c) - start(c))$$

measuring the total delay of output jobs in the schedule σ *in regards of the desired start times.*

Example 1 Here is an example (Fig. 1) of a problem instance P_1 with four devices $D = \{L_1, L_2, L_3, L_4\}$, 20 output jobs and the following desired start times:

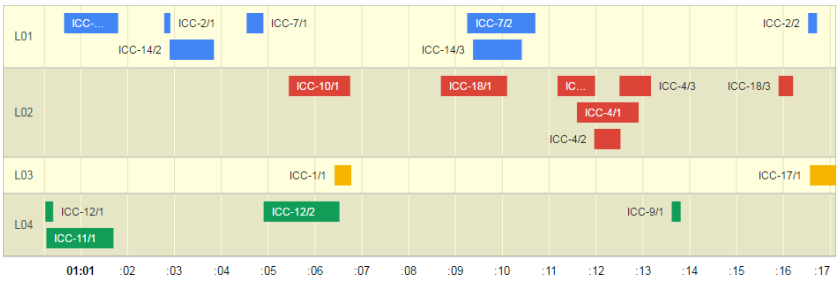

Figure 1: An example problem P_1

That problem instance P directly defines also a schedule σ_0 based on the desired start times that is correct, if no devices in D are incompatible, i.e. $I_0 = \emptyset$. This schedule however is not correct anymore, if we consider the incompatibility relation I_1 in which every device $L \in D$ is incompatible with itself. A correct schedule σ_1 for this incompatibility relation is shown in Figure 2.

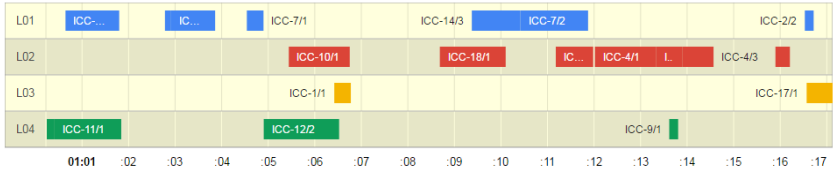

Figure 2: A correct schedule for the problem instance P_1 and incompatiblity relation I_1

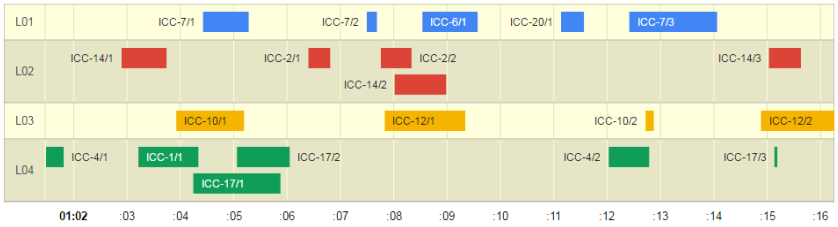

Figure 3: An example problem P_2

Example 2 Fig. 3 is an example of a problem instance P_2 with four devices $D = \{L_1, L_2, L_3, L_4\}$, 20 output jobs and their desired start times.

Consider the incompatiblity relation I_2 that defines that any two devices that are adjacent to each other in the scheduling table shown in Fig. 3 are incompatible with each other, e.g. L_3 is incompatible with all of $\{L_2, L_3, L_4\}$, but L_1 is not incompatible with L_4 for instance. A correct schedule σ_2 for P_2 with incompatiblity relation I_2 is shown in Figure 4.

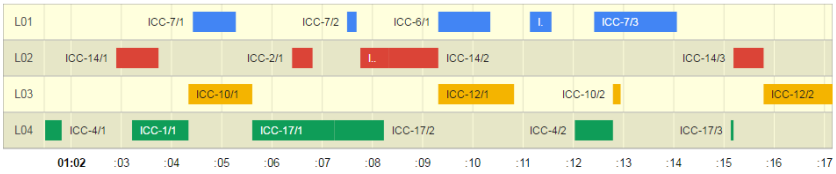

Figure 4: A correct schedule for the problem instance P_2 and incompatiblity relation I_2

The respective decision problem to the optimization problem can be defined as follow:

Definition 2 (OUTPUT JOB SCHEDULING Decision Problem)**.** *Given a finite set $C = \{c_1, \ldots, c_n\}$ of output jobs, a finite set of device names $D = \{d_1, \ldots, d_k\}$, total functions*

- *$start : C \to \mathbb{N}_0$ for desired start times,*
- *$duration : C \to \mathbb{N}_0$ for the expected output duration,*
- *$device : C \to D$ for the target output device*

and an incompatibility relation $I \subseteq D \times D$, and a constant $q \in \mathbb{N}_0$, find a total function $\sigma : C \to \mathbb{N}_0$ called schedule *that satisfies all of the following (correctness) constraints:*

- *$start(c) \leq \sigma(c)$ for all $c \in C$, and*
- *for all $c_1, c_2 \in C$ whenever $c_1 \neq c_2$ and $I(device(c_1), device(c_2))$ then c_1 and c_2 are non-overlapping in σ, i.e. $\sigma(c_1)+duration(c_1) \leq \sigma(c_2)$ or $\sigma(c_2)+duration(c_2) \leq \sigma(c_1)$*

and the solution quality constraint:

$$delay(\sigma) = \sum_{c \in C} (\sigma(c) - start(c)) \leq q$$

ensuring that the total delay of output jobs in the schedule σ in regards of the desired start times does not exceed the given bound q.

Note, that a correct schedule to the output job scheduling problem always exists: simply schedule all contexts one after the other (in some arbitrary order) such that at any point in time at most one output job is being active and all output jobs are being scheduled only after the desired start time. For later, we observe that the special case where all desired start times are 0 and the expected output duration is 1 any permutation of C defines a correct schedule of length n when scheduling the i-th element in the permuation at time i.

2 Computational Complexity

In this section, we investigate the computational complexity of the output job scheduling problem (see Def. 1). We will show that the optimization version of the problem is **NP**-hard [1] by constructing a polynomial time reduction from INDENDENT SET to OUTPUT JOB SCHEDULING. We also show that the decision problem is **NP**-complete by a polynomial-time reduction of the optimization version of that problem.

Definition 3 (INDEPENDENT SET)**.** *Given a finite undirected graph $G = (V, E)$ and some integer $k \leq |V|$ determine if there is an* independent set $S \subseteq V$ *of size $|S| \geq k$, i.e. a set S such that for all $v_1, v_2 \in V$ there is no edge in E that connects v_1 and v_2.*

Lemma 1. INDEPENDENT SET *is* **NP***-complete.*

[1, 2] both give a proof of **NP**-completeness of INDEPENDENT SET. The proof in [2] constructs a polynomial-time reduction of 3-SAT to INDEPENDENT SET.

Theorem 1. *The* OUTPUT JOB SCHEDULING *optimization problem is* **NP***-hard.*

Proof. We will proof the theorem by constructing a polynomial time reduction from INDEPENDENT SET to OUTPUT JOB SCHEDULING: Let $P = (G, k)$ be instance of INDEPENDENT SET with a graph $G = (V, E)$ consisting of vertices V and edges E.

Based on P we construct a corresponding instance P^* of the optimization version of OUTPUT JOB SCHEDULING by setting

- the set of output jobs $C := V$
- the set of device names $D := V$
- the desired start times $start(c) := 0$ for all $c \in C$,
- the he expected output duration $duration(c) := 1$ for all $c \in C$
- the target output device $device(c) := c$ for all $c \in C$
- and the incompatibility relation $I := E$

The mapping from P to P^* takes linear time in the size of P. We now show that the mapping is also a reduction from INDENDENT SET to OUTPUT JOB SCHEDULING, i.e. it satisifes the following property $(*)$:

P contains an indenpendent set S of size at least k if and only if an optimal solution σ^* to P^* schedules the output of at least k output jobs in parallel.

First, we proof the implication from right to left (,,if"): Let σ^* be an optimal solution to P^* that schedules the output of at least k output jobs in parallel, i.e. there exists some $t \in \{0, \ldots, |V - 1|\}$ such that $parallel(\sigma^*, t) := \{c \in C \,|\, start(c) = t \text{ in } \sigma^*\}$ has at least size k. Since any two different elements in $c, c' \in parallel(\sigma^*, t)$ overlap with each other in σ^* and this schedule is correct for P^* it must hold that for any two different elements in $c, c' \in parallel(t)$ the respective devices $device(c) = c$ and $device(c') = c'$ are not incompatible in I, hence c and c' are not connected in E. Therefore, $parallel(\sigma^*, t)$ is an independent set in G of size at least k.

Next, we proof the implication from left to right (,,only if"): Let S be an independent set in G with size at least k and σ^* be an optimal solution for P^*.

We can observe the following: for *any optimal* solution σ of P^* it must hold that

1. All sets $parallel(\sigma, t)$ are independent sets in G for $t \in \{0, \ldots, |V| - 1\}$

2. All sets $parallel(\sigma, t)$ are either maximal independent sets in G or for all $c \in C$ such that $parallel(\sigma, t) \subset parallel(\sigma, t) \cup \{c\}$ is an independent set in G there exists a $t' < t$ such that $c \in parallel(\sigma, t')$ (i.e. either they are maximal , or all possible output jobs that would cause an extension of the set to a maximum independent set in G are already scheduled earlier in σ).
3. The set $parallel(\sigma, 0)$ is a maximal independent set in G
4. For any $t, t' \in \{0, \ldots, |V| - 1\}$ such that $t \leq t'$: $parallel(\sigma, t)$ contains at least as many elements as $parallel(\sigma, t')$. Thus, the sequence of sizes of the output jobs scheduled at time $t = 0, 1, \ldots$ is monotonically decreasing.
5. The set $parallel(\sigma, 0)$ is a maximum independent set in G

The first item holds because σ^* is a correct schedule wrt. the incompatiblity relation I defined in P^* (applying the same argument as above in the ,,if" part of property $(*)$).

To show the second item, we know because of the first item that all $parallel(\sigma, t)$ are independent sets for all t. If $parallel(\sigma, t)$ is maximal, the statement holds. So lets consider the case that $parallel(\sigma, t)$ is not maximal in G. Then, there exists $c \in C$ such that $parallel(\sigma, t) \subset parallel(\sigma, t) \cup \{c\}$ is an independent set in G. Consider any such $c \in C$. Since c is scheduled in σ, c has to be scheduled either before or after t. If it would be scheduled at time t' later than t in σ, then σ would not be an optimal solution since we could strictly reduce the total delay by moving c in σ to start at time $t < t'$. Thefore, any such c has to be scheduled strictly before t in σ. Thus, the statement of the second item holds.

The third item follows immediately from the second item.

The fourth item is also simple to show: assume that for some $t < t'$ we would have $parallel(\sigma, t)$ contains less elements than $parallel(\sigma, t')$. Then we could simply construct a schedule σ' that switches the output jobs to be output at times t and t' and thus we would have a σ' with $parallel(\sigma', t) = parallel(\sigma, t')$ and $parallel(\sigma', t) = parallel(\sigma, t')$. Since then $parallel(\sigma', t)$ is smaller than $parallel(\sigma', t')$ and otherwise σ and σ' are the same, we constructed a schedule σ' that has a smaller total delay $delay(\sigma') < delay(\sigma)$ than σ, which contradicts that σ is an optimal solution (wrt. the total delay). Therefore, the assumption must be wrong and the statement of item four holds.

It remains to show the last item: Assume that $parallel(\sigma, 0)$ is not a maximum independent set in G. Because of the third item above, we know that $parallel(\sigma, 0)$ is a maximal independent set. Hence, there exists a maximal independent set M in G that has more elements than $parallel(\sigma, 0)$. So, in schedule σ a smaller set than M is scheduled at time 0. and $|M| > |parallel(\sigma, 0)|$. With the fourth item above, we also have $|M| > |parallel(\sigma, t)|$ for all $t \in \{0, \ldots, |V| - 1\}$. Hence, the maximum indendent set M cannot be scheduled in σ as one set, but it is partitioned into parts $M_0, \ldots, M_{|V|-1}$ (some of which may be empty) such that $|M_t| < |parallel(\sigma, 0)|$ and $\cup_t M_t = M$ and $M_t \cap M_{t'} = \emptyset$ for all $t, t \in \{0, \ldots, |V| - 1\}$ and $t \neq t'$.

We construct a schedule σ' from σ as follows: Select the smallest $j > 0$ such that M_j is a non-empty partition of M. Now, schedule all output jobs that are scheduled at time 0 in σ at time j in σ'. Then, schedule alle output jobs in M at time 0 in σ'. We schedule at time $t \geq 1, t \neq j$ in σ' all output jobs that are scheduled at time t in σ and that are not also elements in M_t.

The new schedule σ' schedules all output jobs that have been scheduled by σ and it is a correct schedule for P^* too. Further, it has a smaller total delay than σ: Since $|M| > |parallel(\sigma, 0)|$ we schedule now strictly more output jobs at time 0, i.e. $|M| = |parallel(\sigma', 0)| > |parallel(\sigma, 0)|$. Compared to σ, we delayed $|parallel(\sigma, 0)|$ by j time units, so we increase total delay in σ' compared to σ by $j \cdot |parallel(\sigma, 0)|$. At the same time, we decrease the total delay by $|M| \cdot j$ because (i) $\cup_j M_j = M$, (ii) we schedule all elements of M at time 0, and (iii) all of these elements have been scheduled at time $t \geq j > 0$ in σ.

The difference $delay(\sigma') - delay(\sigma) = j \cdot (|parallel(\sigma, 0)| - |M|) < 0$ is smaller than zero and therefore, σ' has a smaller total delay than σ. This contradicts that σ is optimal. Hence, our assumption is wrong and $parallel(\sigma, 0)$ is a maximum independent set in G.

Using these properties of optimal solutions, we can complete the proof of the implication by showing that in σ^* there exists a $parallel(\sigma^*, t)$ such that $k \leq |S| \leq |parallel(\sigma^*, t)|$: we know that $parallel(\sigma^*, 0)$ is a maximum independent set in G, hence it is at least as big as the independent set S and therefore $k \leq |S| \leq |parallel(\sigma^*, t)|$ for $t = 0$.

Using property $(*)$ we can then construct an algorithm to reduce INDEPENDENT SET in polynomial time and correctly decide the original problem for all problem instances based on an optimal output job scheduling algorithm: Given P we compute P^* in polynomial time, then compute an optimal solution σ^* to the corresponding information scheduling problem P^*. Then we give an answer to the original problem: we return "yes" for P if and only if the computed optimal solution σ^* for P^* schedules the output of at least k output jobs in parallel. The latter requires to compute $parallel(\sigma^*, 0) = \{c \in C \mid start(c) = 0 \ in \ \sigma\}$ and test if the set contains at least k output jobs. This takes at most $O(n)$ many additional steps. So overall, we have construct a polynomial time reduction that we aimed for. Because of Lemma 1, the optimal output job scheduling problem is therefore **NP**-hard. □

Observations. In fact, the proof shows that the optimization problem is already **NP**-hard when we do not consider

- individual release times for output jobs (i.e. all desired start times are the same, e.g. 0, hence no specific individual constraint to meet for the scheduler for each output job)
- individual expected output durations for the output jobs (i.e. when we consider only output jobs with unit output length)

Further, the incompatibility constraints I are a crucial source of complexity: If we consider $I = \emptyset$, the problem becomes trivial and can be solved in linear time: simply output schedule σ_0 as defined by the individual desired output dates for output jobs, since this schedule has $delay(\sigma_0) = 0$ and thus clearly is an optimal solution.

Based on Theorem 1, we can proof the **NP**-hardness of the respective decision problem:

Lemma 2. *The* OUTPUT JOB SCHEDULING *decision problem is* **NP***-hard*

Proof. We construct a polynomial-time reduction of the OUTPUT JOB SCHEDULING optimization problem to the OUTPUT JOB SCHEDULING decision problem:

We can always construct a correct and complete schedule for a problem instance P of the OUTPUT JOB SCHEDULING problem by scheduling output jobs sequentially, i.e. one after another such that no two output jobs overlap in time and between any two output jobs that have been scheduled, there is not gap between them. Thus, the maximal possible total delay for any correct schedule is at most

$$D = n(max_{c \in C} start(c) + (n-1) max_{c \in C} duration(c))$$

since for each of the n output jobs that are scheduled, there are at most $n - 1$ output jobs scheduled before, each can cause at most a delay of $max_{c \in C} duration(c)$ and we safely begin to schedule all output jobs after the latest desired start time to satisfy all start time constraints in P.

Thus, there are only finitely many possible upper bounds q on the quality of correct schedule σ to consider, i.e. $q \in \{0, \ldots, D\}$. We can perform binary search on $q \in \{0, \ldots, D\}$ using the algorithm to solve the OUTPUT JOB SCHEDULING decision problem for the given problem instance P with quality bound q in order to find the smallest value q^* such that

there exists a correct schedule σ^* for P that satsfies the quality bound q^* in $O(log(D))$ search steps. This is possible because of the monotonicity of the decision problem: if there is a correct schedule σ with a total delay $delay(\sigma) \leq q$, then we can infer that the same holds for all $q' \geq q$; on the other hand, if there is no correct schedule σ with a total delay $delay(\sigma) \leq q$, then we can infer that the same holds for all $q' \leq q$ too.

This allows us to compute an optimal schedule σ^* in at most polynomialy many calls (in regards of the size of the input problem P) to the decision procedue for the OUTPUT JOB SCHEDULING decision problem. Therefore, we have a polynomial time reduction of the optimization problem to the decision problem. Since, the optimization problem is **NP**-hard, the decision problem is **NP**-hard too. □

Lemma 3. OUTPUT JOB SCHEDULING *decision problem is in* **NP**.

Proof. Given an instance P of the OUTPUT JOB SCHEDULING decision problem with a quality bound q, we can observe:

For all $c \in C$ the candidate sets for $\sigma(c)$ are subsets of $\{0, \ldots, D\}$ for the value D defined in the proof of Lemma 2 above and therefore finite. Hence there are only finitely many candidate schedules σ to consider. Given a candidate schedule σ, we can verify the conditions for correctness of the schedule from Def. 1 in polynomial time with respect the problem instance size (assuming unary coding of numbers). Therefore, we can define a non-deterministic Turing machine that first guesses non-deterministically a candidate schedule σ and then checks in polynomial time for correctness of the schedule and if the upper bound on the solution quality q is met by the schedule. If that is the case, the machine stops and accepts the input P. Otherwise, the machine does not accept the input. □

Given Lemma 2 and Lemma 3 , we have shown

Theorem 2. *The* OUTPUT JOB SCHEDULING *decision problem is* **NP**-*complete.*

Corollary 1. *Any restricted class $\mathcal{G}$ of graphs for which* INDEPENDET SET *is* **NP**-*hard, the* OUTPUT JOB SCHEDULING *is also* **NP**-*hard when restricting the device incompatibility relation to graphs in $\mathcal{G}$.*

Proof. Our proof of Theorem 1 and Lemma 2 constructs a reduction that does not change the input graph of the INDEPENDENT SET instance considered. Hence, it applies in the same way for any restricted class of graphs for which INDEPENDENT SET is **NP**-hard. □

This allows us to directly identify many more restricted cases, where the output job scheduling problem already is **NP**-hard [1], for instance cubic planar graphs, total graphs of bipartite graphs, and graphs containg no triangles.

References

[1] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[2] Christos M. Papadimitriou. *Computational complexity.* Addison-Wesley, Reading, Massachusetts, 1994.