11-2020

What If You Are Late on Several (Relatively Small) Tasks?

Olga Kosheleva
The University of Texas at El Paso, olgak@utep.edu

Vladik Kreinovich
The University of Texas at El Paso, vladik@utep.edu

Comments:

Technical Report: UTEP-CS-20-111

Published in *International Mathematical Forum*, 2020, Vol. 15, No. 8, pp. 401-406.

What If You Are Late on Several (Relatively Small) Tasks?

Olga Kosheleva and Vladik Kreinovich
University of Texas at El Paso
El Paso, TX 79968, USA
olgak@utep.edu, vladik@utep.edu

Abstract

In practice, we sometimes end up in a situation when we are late on several relatively small tasks. We cannot finish them all, so which ones should we do first? We show that in general, this is an NP-complete problem. In the typical situation when all the tasks are of approximately the same importance and requires approximately the same time to finish, we can have an explicit solution to this problem. In a nutshell, the resulting (somewhat counterintuitive) recommendation is to start with things which are not yet late or only a few days late. Actually, this recommendation makes sense: on a task on which we are already 30 days late, making it 31 days late will not change much, but if a task is due today, we want to finish it today, to avoid late penalty.

1 Formulation of the Problem

Practical situation. Many of us found ourselves in a situation when we are already late on several relatively small tasks:

- there many be several emails to which we did not yet answer;
- there may be several papers that we promised to review – and for which the deadline has passed;
- there may be several student assignment that we still need to grade – and the promised deadline has passed; and
- there may be all of this: emails to reply, papers to review, assignments to grade, etc.

Resulting problem. In such a situation, a natural question is: which of the late tasks should we start with? Or should we first finish the tasks which are due today?

What we do in this paper. In this paper, we analyze how to solve this problem.

For this purpose:

- first, we formulate this problem in precise terms, and
- then we analyze how to solve the resulting precisely formulated problem.

2 Describing the Problem in Precise Terms

What is known. Let n be a number of relatively small tasks on which we are already late – or which are due today. For each such task i ($i = 1, \ldots, n$):

- Let t_i denote the number of days past promised deadline at the start of today's working day, i.e., the difference between today's date and the promised deadline. In particular, for tasks which are due today, the promised deadline is today, so $t_i = 0$.
- Let d_i denote the time (in days) that we need to finish this task. We consider relatively small tasks, each of which can be performed in a working day, so $d_i \le 1$.

For each of these tasks, there is a penalty of being late. Let $p_i(t)$ denote the (accumulated) penalty of being t days late in task i.

Comment. There is no penalty for being on time, so $p_i(0) = 0$ – and, of course, $p_i(t) = 0$ for $t < 0$: there is no penalty if we finish the taks before the deadline.

What we want. We want to minimize the resulting accumulated overall penalty at the end of the working day. For this, we need to decide how much effort $e_i \ge 0$ to allocate today to each of the n tasks. In particular, if we do not spend any time today on the i-th task, this means $e_i = 0$. We are considering one working day, so the overall amount of effort is 1: $\sum\limits_{i=1}^n e_i = 1$.

For each task i:

- If we finish the i-th task today, i.e., if $e_i = d_i$, then the penalty will be exactly $p_i(t_i)$.
- If we do not finish the i-th task today, i.e., if $e_i < d_i$, then the next day we will be one more day late, so the penalty will grow to $p_i(t_i + 1)$.

Resulting formulation of the problem in precise terms. We are given:

- the values $t_i \ge 0$ and $d_i \in [0, 1]$, $i = 1, \ldots, n$, and
- n non-strictly increasing functions $p_i(t)$ for which $p_i(0) = 0$.

Among all the tuples $(e_1, \ldots, e_n)$ for which $0 \le e_i \le d_i$ for all i and

$$\sum_{i=1}^n e_i = 1,$$

we need to find the tuple that minimizes the resulting penalty

$$P \stackrel{\text{def}}{=} \sum_{i:e_i=d_i} p_i(t_i) + \sum_{i:e_i<d_i} p_i(t_i+1) \tag{1}$$

attains its smallest possible value.

3 How Can We Solve This Optimization Problem

Let us simplify the above formulation. To simplify the above formulation, let us reformulate the expression (1) as follows:

$$P = \sum_{i=1}^n p_i(t_i+1) - \sum_{i:e_i=d_i} \Delta_i, \tag{2}$$

where we denoted

$$\Delta_i \stackrel{\text{def}}{=} p_i(t_i+1) - p_i(t_i). \tag{3}$$

The first term in the right-hand side of the formula (2) does not depend on us, so minimizing the expression (2) is equivalent to maximizing the sum

$$\sum_{i:e_i=d_i} \Delta_i. \tag{4}$$

To simplify the problem further, let us introduce a binary (0-1) variable x_i for which:

- $x_i = 1$ if we finish the i-th task today, i.e., if $e_i = d_i$, and
- $x_i = 0$ if we do not finish the i-th task today, i.e., if $e_i < d_i$.

In these terms, the above optimization problem takes the following form.

Resulting formulation. We are given:

- the values $d_1, \ldots, d_n \in [0, 1]$ and
- the values $\Delta_1, \ldots, \Delta_n \ge 0$.

Among all the tuples $(x_1, \ldots, x_n)$ for which $x_i \in \{0, 1\}$ for each i and

$$\sum_{i=1}^n x_i \cdot d_i \le 1,$$

we need to find the tuple for which the sum

$$\sum_{i=1}^{n} x_i \cdot \Delta_i$$

attains its largest possible value.

This is exactly the knapsack problem. From the computational viewpoint, this is exactly the well-known knapsack problem:

- we have n objects, we know their weights $w_1, \ldots, w_n$ and their prices $r_1, \ldots, r_n$, and we know how much weight W a knapsack can carry;
- we want to select which objects to place in the knapsack so as to maximize the overall price.

If we introduce auxiliary variables $x_i \in \{0, 1\}$ so that $x_i = 1$ if we take the i-th object and $x_i = 0$ if we do not, then this problem takes the following form: Among all the tuples $(x_1, \ldots, x_n)$ for which $x_i \in \{0, 1\}$ for each i and

$$\sum_{i=1}^{n} x_i \cdot w_i \le W,$$

we need to find the tuple for which the sum

$$\sum_{i=1}^{n} x_i \cdot r_i$$

attains its largest possible value.

For $w_i = e_i$ and $r_i = \Delta_i$, this is exactly our problem.

This is good news and bad news. This is good news because there are many efficient algorithms for solving the knapsack problem (see, e.g., [1]), so we can use them to decide which tasks to perform first.

This is bad news because it is known that knapsack problem is NP-hard – meaning that unless P = NP (which most computer scientists believe to be impossible) – no feasible algorithm can solve all the instances of this problem.

A frequent case. A frequent case is when all the tasks are of the same importance and require approximately the same amount of time to finish:

$$e_1 = e_2 = \ldots$$

In this case, the limitation

$$\sum_{i=1}^{n} x_i \cdot e_i \le 1$$

simply means that the overall number of tasks that we can perform in a day is limited by the value $m \overset{\text{def}}{=} \dfrac{1}{e_1}$.

So, the question becomes:

- we have n values $\Delta_1, \ldots, \Delta_n$,
- we need to select m of them for which the sum of the corresponding values Δ_i is the largest possible.

One can check that in this case, our problem has a straightforward solution: namely, we can sort the value Δ_i into a decreasing sequence

$$\Delta_{(1)} \geq \Delta_{(2)} \geq \ldots \geq \Delta_{(n)},$$

and we select n tasks with the largest possible values Δ_i, i.e., the values

$$\Delta_{(1)}, \Delta_{(2)}, \ldots, \Delta_{(m)}.$$

Resulting commonsense recommendation. Usually, the increase in penalty slows down:

- we may have penalty for being late one day, but
- if we are already 30 days late, it is highly improbable that the penalty will seriously increase if we become 31 days late.

In this case, the differences $\Delta_i = p_i(t_i + 1) - p_i(t_i)$ decrease as t_i increases.

Thus, the above recommendation means that:

- we concentrate all our efforts on tasks which are due today or at least only a few days late, and
- we only deal with already-seriously-late tasks when we have time left.

Acknowledgments

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science), and HRD-1834620 and HRD-2034030 (CAHSI Includes).

References

[1] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*, Springer, Berlin, Heidelberg, 2010.