Data transfer scheduling for deep space exploration

Emmanuel Hébrard, Christian Artigues, Pierre Lopez, Arnaud Lusson, Steve A. Chien, Gregg R. Rabideau, Adrien Maillard

Data transfer scheduling for deep space exploration

Emmanuel Hebrard* Christian Artigues* Pierre Lopez*
Arnaud Lusson* Steve Chien† Adrien Maillard† Gregg Rabideau†

1 The overlapping memory dumping problem

Scientific instruments for deep space exploration spacecrafts become more and more sophisticated and consequently produce more and more data that must be sent to Earth. Data management is highly critical as the onboard memory is limited and communication with Earth is often a bottleneck. In this paper we consider the same case as in [2] in the context of the Rosetta/Philae mission, where the data production plan is known and the problem consists in planning memory dumps. Data is produced into several memory buffers and the goal is to avoid data loss, which occurs when data are produced into a full buffer. More precisely, we want to maximize the minimum margin of buffer, where the margin is the percentage of its capacity left free. This objective aims at designing robust plans where unexpected changes of the dump or fill rates can be absorbed by the margins. Under this assumption, the fill rate of each memory buffer over the planning horizon is part of the input. Data can only be dumped when the spacecraft is visible from Earth. This is materialized by consecutive disjoint downlink windows. Data dumping is a semi-automatized process. For each downlink window a priority has to be assigned to each buffer, then, the transfers follow this priority ordering. We therefore consider the problem of computing a priority assignment that maximizes the minimum margin.

In the overlapping Memory Dumping Problem (oMDP), we are given m downlink windows, where $[s_j, e_j[$ stands for the time interval in which the downlink j is available, and δ_j for the dump rate for transfers of downlink j. Moreover, there are n memory buffers, where C_i stands for the capacity of buffer i; $r_i(j)$ for the maximum handover (residual) usage of buffer i at the end of downlink window j; and $f_i : \mathbb{R} \mapsto \mathbb{R}^+$ for the piece-wise constant fill rate function of buffer i over time. Let ν be the number of inflection points over all fill rate functions.

The transfers can be controlled by setting a priority function over the buffers defining a *ranking*. At every step, one of the buffers of highest priority among those who have a packet in memory is selected via round-robin to transfer a packet.

*{hebrard,artigues,lopez}@laas.fr. LAAS-CNRS, Université de Toulouse, CNRS, France
†{steve.a.chien,adrien.maillard,gregg.r.rabideau}@jpl.nasa.gov. Jet Propulsion Laboratory, California Institute of Technology, CA, USA

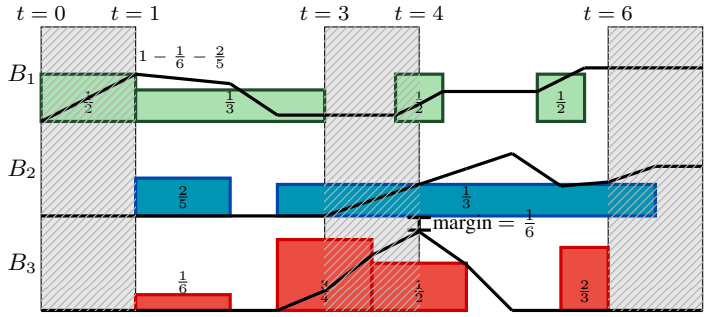

Figure 1: Example with 3 buffers. White areas indicate visibility.

Let $U_i : \mathbb{R} \mapsto \mathbb{R}$ be the quantity of data on buffer i over time and $g_i : \mathbb{R} \mapsto \mathbb{R}$ be the transfer rate out of buffer i over time.

In between downlink window, all transfer rates are null and the memory usage growth is the fill function, for every buffer i. In particular, for any time t preceding the start of the first downlink s_1, we have $g_i(t) = 0$ and $U_i(t) = \int_0^t f_i(x)\,dx$.

During a downlink window with dump rate δ, however, the effective transfer rate at time t depends on the priority function P, and on the usages and fill rates at time t.

We consider the decision problem of finding a priority ranking for every downlink, such that the peak usage of any buffer (given its fill rate functions, and the data transfer system described above) is less than its capacity, i.e., without *data loss*. The objective function is actually to maximize the minimum margin, $\min\{M(i) \mid i \in B\}$, where $M(i)$ is defined as one minus the ratio between the peak usage and the capacity of buffer i. However, it can be achieved by dichotomic search using an algorithm for the decision problem above.

Example 1. *Figure 1 shows a plan with two downlinks, both with dump rate* 1*, and 3 buffers all of capacity* 1*. Fill rates are figured by colored rectangles, e.g.,* $f_1(t) = \frac{1}{2}$ *for* $t \in [0, 1]$*. In the first downlink, all buffers have equal priority* $P_1(1) = P_1(2) = P_1(3) = 1$*, in the second, buffer 3 has the highest priority then buffer 2 then buffer 1. The black curves stand for the resulting buffer usage over time* $U_i(t)$*. The minimum margin is* $\frac{1}{6}$*.*

2 Complexity and algorithms

The software used for the real mission (DALLOC) is described in [2] and relies on a heuristic called DOWNLINKCOUNT. In a nutshell, this method assigns priorities based on when each buffer would exceed a given target margin. More precisely, it counts how many downlink windows would occur before exceeding the target if no data could be dumped. At each downlink window $j \in [1, m]$, the transfers are computed starting from the current usage, and with a dump rate equal to 0. Then a priority assignment is extracted as defined above, and window j is simulated using the same procedure but with dump

rate δ_j. It is important to notice that the choice of target margin can change the priority assignment. Therefore, in DALLOC, this heuristic is used within a binary search. The resulting algorithm is denoted ITERATEDLEVELING.

As the complexity of the problem was not established, we first consider the decision problem PRIORITYALLOC: "Is there a priority assignment for each downlink window such that there is no data loss?".

First, we show that the problem is in NP because, given a priority assignment, simulating the transfers and therefore verifying a certificate, can be done in polynomial time, using similar arguments as in [1]. This is not trivial to see that not every packet of data need to be tracked individually, and the number of packets can be exponential in the size of the instance. Moreover, to compute the usage over time of a given buffer, we only need to know the set of buffers of strictly higher priority and the set of buffers of equal priority (Property 1). In other words, the exact priority assignment within the set with higher priority is irrelevant. We define a procedure (denoted Algorithm 1 from now on) to compute the memory usages (and so the minimum margin) in $O((\nu+m)(\log(\nu+m)+n^2\log n))$ time, given a priority assignment P on each downlink window. Then we use a reduction from partition to show that PRIORITYALLOC is NP-complete.

Next we consider the single-window problem ($m = 1$). Thanks to Property 1 and Algorithm 1, it is possible to compute the margin $M_{\Gamma\prec\Omega}(i,j)$ of buffer i in window j when the buffers in the set Γ have a strictly higher priority than i, and the buffers in the set Ω have the same priority as i. We propose a method (Algorithm 2) to solve PRIORITYALLOC on a single downlink window with $O(n^2)$ calls to Algorithm 1.

Finally, for the general case, we propose a heuristic (REPAIRDESCENT) which implements a greedy "descent": a seed priority assignment is computed using the heuristic DOWNLINKCOUNT and iteratively "repaired" to achieve a strictly higher target margin.

The repair procedure runs the current priority assignment using Algorithm 1 until reaching downlink j at which the target margin is exceeded. Then, it calls Algorithm 2 to check whether there exists a priority assignment P_j that achieves the expected margin. If there is such an assignment, it is run by Algorithm 1 and we advance to downlink window $j+1$. If the last window is reached, an improving solution has been found. Otherwise, if downlink j does not have a priority assignment without data loss, Algorithm 2 returns a set B of buffers guaranteed to exceed their target margin given the current usage at the end of downlink $j-1$, even if they are given (globally) the highest priority. Therefore, the only way to avoid data loss in this downlink is to reduce the residual load for at least one of these buffers at the end the previous downlink. In order to avoid branching on the possible ways of reducing this load, we instead add one randomly chosen handover constraint. We pick a random buffer i in the set B, and we set its maximum handover value $r_i(j-1)$ to the current usage $U_i(j-1)$ to which we subtract the gap we need to achieve the expected margin: the margin we obtain when buffer i is given highest priority with all other buffers in B (that is, $M_{\emptyset\prec B}(i,j)$) minus the expected margin obj. Then the previous window is solved again with this new constraint: it "backtracks" by decreasing the current downlink window j. If the current downlink window is the first ($j = 1$), then no such constraint can be posted and we fail.

3 Experimental Evaluation

We used the same data set as in [2], constituted of four scenarios (`MTP1`, `MTP2`, `MTP3` and `MTP4`), corresponding to the whole activity sequence of Rosetta divided in four quarters. The whole sequence has over 40,000 data production events for 16 memory buffers and 324 downlink windows. Its duration is about three months and a half with a precision of one second. Table 1 give the margin value DOWNLINKCOUNT incorporating Algorithm 1 and DALLOC's published results [2], as well as average CPU times for both algorithms. We observe that our reimplementation is much faster than the original one.[1]

	Margin				Avg. CPU
	`MTP1`	`MTP2`	`MTP3`	`MTP4`	
DALLOC's	40.4	46.5	17.8	29.8	14.6
Ours	46.4	68.1	17.8	30.8	0.018

Table 1: New and old implementations of DOWNLINKCOUNT.

Moreover, REPAIRDESCENT finds a more robust download plan for the hardest real scenario (`MTP4`). Both methods match an upper bound on `MTP1`, `MTP2`, and `MTP3`.

	Margin				Avg. CPU
	`MTP1`	`MTP2`	`MTP3`	`MTP4`	
Upper Bound	46.4	72.5	54.8	53.4	
ITERATEDLEVELING	46.4	72.5	54.8	48.5	0.116
REPAIRDESCENT	46.4	72.5	54.8	**52.8**	0.088

Table 2: Comparison with the state of the art on real scenarios.

Finally, on a much larger randomly generated data set, REPAIRDESCENT is consistently faster than ITERATEDLEVELING to find solutions of similar quality and is able to significantly improve the minimum margin when given more CPU time.

References

[1] G. SIMONIN, C. ARTIGUES, E. HEBRARD AND P. LOPEZ (2015). Scheduling scientific experiments for comet exploration. *Constraints: An International Journal* 20(1), 77–99.

[2] G. RABIDEAU, S. CHIEN, M. GALER, F. NESPOLI AND M. COSTA SITJÀ (2017). Managing Spacecraft Memory Buffers with Concurrent Data Collection and Downlink. *Journal of Aerospace Information Systems* 14 (12), 637–651.

[1] The slight difference in margin values is because DALLOC's results include pre-assigned priorities. We did not have the data to reproduce it.