A Mathematical Programming Solution to the Mars Express Memory Dumping Problem

Giovanni Righini and **Emanuele Tresoldi**
Dipartimento di Tecnologie dell'Informazione
Università degli Studi di Milano
Via Bramante 65, 26013 Crema, Italy

Abstract

The memory dumping problem arises in the context of planning and scheduling activities of the Mars Express mission of the European Space Agency. The problem consists of scheduling scientific data transmission from Mars to the Earth. A previously developed algorithm computes robust schedules in a heuristic way and iteratively improves the schedule robustness by solving a sequence of max-flow problems. We present a linear programming algorithm to compute schedules of maximum robustness, providing provably optimal solutions in a very short computing time. We also give necessary and sufficient conditions to characterize "easy" and "difficult" instances, such that the former ones can be solved directly without any optimization algorithm.

Introduction

The Mars Express mission of the European Space Agency aims at the observation of Mars from a satellite orbiting around around the Red Planet. The satellite is equipped with many different devices to gather scientific data about Mars. These data are kept in a memory device on board and they are periodically transmitted to Earth stations during suitable visibility time windows.

The Mars Express memory dumping problem, introduced by (Cesta et al. 2002) and (Oddi et al. 2003), consists of scheduling scientific data transmission from Mars to the Earth.

The problem has been studied in several versions: a simplified version was initially considered in (Oddi and Policella 2004; 2007). The objective is to compute a dumping schedule of maximum robustness in order to cope with uncertainty in the amount of incoming scientific data. The algorithm devised by (Oddi and Policella 2004; 2007) to compute robust schedules is heuristic and iteratively improves the schedule robustness by solving a sequence of max-flow problems with classical polynomial-time specialized algorithms, such as the Ford and Fulkerson algorithm (Ahuja, Magnanti, and Orlin 1993).

In this paper we further elaborate on the formulation presented by (Oddi and Policella 2004; 2007) and we present a linear programming algorithm to compute schedules of maximum robustness. The algorithm provides provably optimal solutions in a very short time. We also give necessary and sufficient conditions to characterize "easy" and "difficult" instances, such that the former ones can be solved directly without any optimization algorithm.

We also describe how a mathematical programming approach can be used to take into account other details of the real application problem, not considered in this simplified version.

Problem definition

Following (Oddi and Policella 2004; 2007) we define the Mars Express memory dumping problem as follows.

Data. We are given a set $\mathcal{I}$ of packet stores with a finite capacity c_i for each $i \in \mathcal{I}$ and a set $\mathcal{J}$ of time windows that are available for the transmission of data to the Earth: the overall available capacity b_j is known for each time window $j \in \mathcal{J}$. In each time period between two consecutive time windows, say $j-1$ and j, scientific data are acquired; these data are stored into the packet stores at the end of each time window $j \in \mathcal{J}$, that is after the transmission of old data to the Earth: the amount d_{ij} of data stored in each packet store $i \in \mathcal{I}$ at the end of each time window $j \in \mathcal{J}$ is known. We remark that the store operation takes effect only at the end of the time window: hence the amount of data d_{ij} cannot be transmitted during time window $j \in \mathcal{J}$. We are also given an initial amount of data d_{i0} in each packet store $i \in \mathcal{I}$.

Variables. The decision variables are the amounts of data to be transferred from each packet store $i \in \mathcal{I}$ to the Earth in each time window $j \in \mathcal{J}$. We indicate these continuous non-negative variables by x_{ij}. We also introduce auxiliary variables y_{ij} to indicate the amount of data stored in each packet store $i \in \mathcal{I}$ after each time window $j \in \mathcal{J}$ and z_{ij} to indicate the amount of data stored in each packet store $i \in \mathcal{I}$ after the data transmission in time window $j \in \mathcal{J}$ and before the store operation occurring in the same time window. All these variables are continuous and non-negative.

Constraints. Flow conservation constraints represent the operations in each packet store $i \in \mathcal{I}$ and in each time window $j \in \mathcal{J}$, as follows:

$$y_{i\,j-1} = x_{ij} + z_{ij} \quad \forall i \in \mathcal{I},\ \forall j \in \mathcal{J} \tag{1}$$

$$z_{ij} + d_{ij} = y_{ij} \ \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \quad (2)$$

To forbid *overdumping* and *overwriting*, upper and lower bounds are imposed to the variables:

$$z_{ij} \geq 0 \ \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \quad (3)$$

$$y_{ij} \leq c_i \ \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \quad (4)$$

Finally the capacity constraints on transmissions are imposed through the following constraints:

$$\sum_{i \in \mathcal{I}} x_{ij} \leq b_j \ \forall j \in \mathcal{J} \quad (5)$$

It is possible to impose that the schedule dump all data to the Earth within the given set of time windows, by setting $y_{i|\mathcal{J}|} = 0$ for all packet stores $i \in \mathcal{I}$.

Objective function. The objective of the optimization is the robustness of the schedule. This is measured by the maximum fraction of capacity of a packet store which happens to be taken by stored data at any point in the schedule. A schedule is robust when this fraction is low, because this yields safety margins to manage unexpected peaks of acquired data. Since the objective function is "min max", we introduce an additional continuous non-negative variable α to be minimized and we replace constraints (4) with the following constraints:

$$y_{ij} \leq \alpha c_i \ \forall i \in \mathcal{I}, \forall j \in \mathcal{J}. \quad (6)$$

With the notation above the problem can be formulated as follows:

$$\min \alpha \quad (7)$$

$$\text{s.t. } y_{i\,j-1} = x_{ij} + z_{ij} \qquad \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \quad (8)$$

$$z_{ij} + d_{ij} = y_{ij} \qquad \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \quad (9)$$

$$y_{ij} \leq \alpha c_i \qquad \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \quad (10)$$

$$\sum_{i \in \mathcal{I}} x_{ij} \leq b_j \qquad \forall j \in \mathcal{J} \quad (11)$$

$$x_{ij} \geq 0 \qquad \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \quad (12)$$

$$z_{ij} \geq 0 \qquad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}. \quad (13)$$

The model yields a linear programming problem, easily solvable by the very effective LP solvers available today, with the additional advantage of the optimality guarantee, not provided by heuristics.

Optimally balanced solutions

The Mars Express memory dumping problem in its simplified version is a linear programming problem and it is polynomially solvable. Owing to the "min max" objective function, the most robust dumping schedule corresponds to an optimally balanced solution, that is a solution in which the amount of data in the packet stores is distributed in an equitable way. This observation leads to a question: "What makes an instance more difficult than another?". To answer this question we have developed a classification criterion to distinguish between "easy" and "difficult" instances, and a fast method to solve easy instances exactly. The same method can also be used to solve difficult instances approximately.

If in each time period the amount of new data assigned to each packet store is roughly proportional to its capacity, it is possible to spread the data among all the packet stores, so that their saturation level is uniform; in this way it is possible to devise a method that yields the optimal solution to the maximum robustness scheduling problem for each time period. Difficult instances of the problem occur when the amounts of new data are not uniformly distributed among all packet stores in each time period. This intuitive concept can be expressed in a formal way as follows.

First of all we define the *saturation level* of a packet store as the ratio between the amount of data stored in it and its capacity. We also define a solution to be *optimally balanced* after a given time window $j \in \mathcal{J}$ if both these conditions hold:

- *Condition 1*: the overall amount of stored data is minimum;
- *Condition 2*: the amount of data stored in each packet store is directly proportional to the capacity of the packet store.

The reason for Condition 1 is that we consider only schedules in which data are transmitted to the Earth as soon as possible, so that no transmission capacity is left unused unless all packet stores are empty. It is easy to prove that leaving more data than necessary in some packet stores cannot improve the robustness of the schedule. Condition 2 is equivalent to asking for a structure of the optimal solution which would be achieved if it were possible to shift data from one packet store to another. In our model this is forbidden by constraints (12): without these constraints one could fill data into a packet store i ($x_{ij} < 0$), so enlarging the available capacity (constraints 11) for dumping data from other packet stores in the same time window; the net effect would be to shift data from other packet stores into packet store i. In the remainder we give necessary and sufficient conditions to characterize "easy" and "difficult" instances; informally, easy instances are such that after each time window it is possible to achieve the same optimal solution as if constraints (12) were not present.

Keeping solutions balanced

Considering two consecutive time windows, say j and $j+1$, we assume that after time window j the solution is optimally balanced (the amount of stored data is minimum and the saturation level is the same for all packet stores) and we give necessary and sufficient conditions for the same property to hold after time window $j+1$.

We indicate the amount of data in packet store $i \in \mathcal{I}$ after the former time window with y_i and the same quantity after the latter time window by y'_i. We assume that $\alpha = \frac{y_i}{c_i} \ \forall i \in \mathcal{I}$, i.e. the same fraction of capacity is used in all packet stores at the end of the former time window. We want to obtain $\alpha' = \frac{y'_i}{c_i} \ \forall i \in \mathcal{I}$, i.e. the same property must hold after the end of the latter time window. We indicate by d_i the amount of data sent to each packet store $i \in \mathcal{I}$ at the

end of the latter time window and by x_i the amount of data downloaded from each packet store $i \in \mathcal{I}$ in the latter time window. We also use the following total quantities:

$$D = \sum_{i \in \mathcal{I}} d_i,$$

$$X = \sum_{i \in \mathcal{I}} x_i,$$

$$C = \sum_{i \in \mathcal{I}} c_i,$$

$$Y = \sum_{i \in \mathcal{I}} y_i$$

and

$$Y' = \sum_{i \in \mathcal{I}} y'_i.$$

They indicate respectively the overall amount of new data stored, the overall amount of old data downloaded, the overall capacity of the packet stores, the overall amount of data stored after the former and after the latter time window. Finally we indicate by B the overall available transmission capacity in the latter time window, that is b_{j+1}. Note that $X = \min\{B, Y\}$.

We must distinguish two different cases: in Case 1 we have $Y \leq B$ and $X = Y$, in Case 2 we have $Y > B$ and $X = B$.

Case 1. If $Y \leq B$, in order to fulfill Condition 1 it is necessary to transmit all the content of all packet stores, i.e. $x_i = y_i \ \forall i \in \mathcal{I}$ and $X = Y$. Therefore the new data are the only data stored at the end of the latter time window; hence an optimally balanced solution is achieved if and only if

$$d_i = \frac{c_i}{C} D \ \forall i \in \mathcal{I}.$$

Case 2. If $Y > B$, in order to fulfill Condition 1 we have $X = B$; however in this case some residual data remain in the packet stores and the values of the variables x_i are not automatically determined. Instead, each of them is constrained between 0 and y_i for each packet store.

The following relations hold:

$$\alpha = \frac{Y}{C} \tag{14}$$

$$\alpha' = \frac{Y'}{C} \tag{15}$$

$$y'_i = y_i - x_i + d_i \ \forall i \in \mathcal{I} \tag{16}$$

Starting from an optimally balanced solution, another optimally balanced solution is achieved if and only if the above equations admit a solution such that $0 \leq x_i \leq y_i \, \forall i \in \mathcal{I}$.

From the equations above we obtain:

$$\alpha' = \frac{Y'}{C} = \frac{\sum_{i \in \mathcal{I}} (y_i - x_i + d_i)}{C} = \alpha + \frac{D - B}{C}.$$

Since each packet store must be filled by a fraction of capacity equal to α', we want to have

$$\frac{y'_i}{c_i} = \alpha' \ \forall i \in \mathcal{I}$$

and hence

$$\frac{y_i - x_i + d_i}{c_i} = \alpha + \frac{D - B}{C} \ \forall i \in \mathcal{I}$$

from which we obtain

$$x_i = d_i - \frac{c_i}{C}(D - B) \ \forall i \in \mathcal{I}.$$

Therefore the condition $0 \leq x_i \leq y_i \ \forall i \in \mathcal{I}$ is equivalent to

$$\frac{c_i}{C}(D - B) \leq d_i \leq \frac{c_i}{C}(D - B) + y_i \ \forall i \in \mathcal{I} \tag{17}$$

or equivalently

$$c_i\left(\frac{D}{C} - \frac{B}{C}\right) \leq d_i \leq c_i\left(\frac{D}{C} - \frac{B}{C} + \alpha\right) \ \forall i \in \mathcal{I} \tag{18}$$

or equivalently

$$\alpha' - \alpha \leq \frac{d_i}{c_i} \leq \alpha' \ \forall i \in \mathcal{I}. \tag{19}$$

When these conditions hold, it is possible to determine the optimal values of the x variables directly from (14), (15) and (16) without having recourse to any optimization algorithm.

Balancing an unbalanced solution

Even if for some time window it is impossible to achieve an optimally balanced solution, as defined before, it is however of practical interest to keep the solution as balanced as possible in the subsequent time window. Following the method outlined above for Case 2, it easy to show that an optimally balanced solution can be achieved after a given time window from *any* solution after the previous time window if and only if the following conditions hold:

$$c_i\left(\frac{D}{C} - \frac{B}{C} + \frac{Y}{C} - \frac{y_i}{c_i}\right) \leq d_i \leq c_i\left(\frac{D}{C} - \frac{B}{C} + \frac{Y}{C}\right) \ \forall i \in \mathcal{I}. \tag{20}$$

The condition resembles condition (18) with the only difference that the upper and lower limits to d_i are now increased by a term $c_i(\frac{Y}{C} - \frac{y_i}{c_i})$, which can be positive or negative according to the load unbalance in packet store $i \in \mathcal{I}$ with respect to the average load $\frac{Y}{C}$.

In Case 1, i.e. when $Y \leq B$, the analysis presented in the previous subsection still holds.

A fast heuristic

The analysis presented here above directly suggests the implementation of a very simple and fast heuristic, which iteratively considers the time windows in chronological order, analyzes each of them in order to detect whether the corresponding balancing subproblem is easy or difficult and solves it, in the former case directly, in the latter case by linear programming by solving a problem like (7)-(13) without index j (because it concerns only one time window at a time). If all subproblems, one for each time window, happen to be easy, then the overall problem is also easy and the overall solution computed by this algorithm is guaranteed to be optimal; otherwise there is no optimality guarantee. The advantage of such a heuristic algorithm is to avoid solving

a potentially large linear programming problem, in case of very tight time restrictions (for instance, when real-time replanning is needed).

The pseudo-code of the heuristic is reported in Algorithm 1.

Algorithm 1 A heuristic algorithm.

```
for all j ∈ J do
  if B ≥ Y then
    // Easy iteration: dump everything //
    for all i ∈ I do
      x_ij := y_i j−1
    end for
  else
    if B = 0 then
      // Easy iteration: dump nothing //
      for all i ∈ I do
        x_ij := 0
      end for
    else
      if Condition (18) holds then
        // Easy iteration: achieve optimal balance //
        for all i ∈ I do
          x_ij := d_ij − (c_i/C)(D − B)
        end for
      else
        // Difficult iteration: optimal balance cannot be achieved //
        // Solve model (7)-(13) for the current time window j //
      end if
    end if
  end if
end for
```

Computational results

Computations experiments have been done on the benchmark instances provided by (Oddi and Policella 2004; 2007), available on line. They have been produced from real data and some randomization, as described in the cited references. In particular data-set B5 is made of 9 different instances, each one with up to 13 packet stores and a number of payload operation requests (yielding data store operations) ranging from 15 and 96.

The linear programming instances have been solved by the free LP-solver GLPK (Gnu Linear Programming Kit), available on-line. Computational experiments have been done on a PC with Intel Core2 Duo T7300 CPU (2.0 GHz) and 2 GB RAM.

Table 1 reports the maximum saturation levels obtained with the algorithm of Oddi and Policella (column OP), those obtained with the heuristic described above (column HEUR) and the optimal ones given by the solution of the LP model (7)-(13) (column OPT).

The results given by the heuristic are significantly better than those of the OP algorithm: the average percentage gap from optimality is 5.40% instead of 10.98%.

Instance	OP	HEUR	OPT
1	0.73	0.72	0.68
2	0.76	0.71	0.65
3	0.78	0.68	0.68
4	0.75	0.66	0.59
5	0.73	0.72	0.68
6	0.74	0.72	0.69
7	0.70	0.66	0.59
8	0.59	0.59	0.59
9	0.59	0.59	0.59

Table 1: Maximum saturation level.

Instance	N. packet stores	HEUR	OPT
B5	13	< 0.001	0.01
B5-28	28	0.10	0.70
B5-54	54	0.11	2.32

Table 2: Average computing time (seconds).

These results are obtained at the expense of a negligible computing time, as shown in Table 2.

There results are referred to three benchmarks: the first one, B5, is the same as before; the other two are artificially made by combining together two or more instances from B5, in order to have a larger number of packet stores. In data-set B5 the percentage of "easy" iterations (those for which the heuristic does not call the LP solver) is greater than 75%, although some "difficult" iteration is always encountered.

For the sake of comparison with the state of the art, the computing time reported in (Oddi and Policella 2007) for solving instance B5 in an approximate way is 21.8 seconds on a 1.8 GHz processor; therefore our exact method turns out to be three orders of magnitude faster and our heuristic turns out to be four orders of magnitude faster.

Scalability. To assess the scalability of our method, we also made computational experiments with larger instances. They are reported in Table 3: the columns indicate the name of the instance, the number of days to be covered by the data dumping plan, the number of time windows in the instance, the number of store operations, the observed computing time, and the maximum saturation level obtained. For these instances we used ILOG CPLEX instead of GLPK as an LP solver. From the examination of the results, it is clear that computing time remains quite acceptable also for large scale instances. It does not depend much on the size of the instance (measured by the number of days, the number of time windows and the number of store operations) but rather on the balance of the incoming scientific data.

Extensions

More complete and realistic versions of the Mars Express memory dumping problem consider other features, such as housekeeping data and priorities, and different objective functions, such as the minimization of dump operations, which are conflicting with the minimization of the saturation considered as the main objective function in (Oddi and

Instance	N. Days	N. TWs	N. Store	Time	Satur.
I71-73	3	15	154	0.17	0.48
I81-83	3	15	145	0.21	0.32
I91-93	3	24	159	0.24	0.37
I71-75	5	25	260	0.21	0.48
I81-85	5	27	232	0.23	0.32
I71-77	7	31	340	0.24	0.48
I81-87	7	34	300	0.24	0.32
I91-97	7	44	352	0.32	0.54
I81-90	10	47	442	0.63	0.32
I81-91	11	56	483	0.82	0.32
I88-98	11	59	522	0.65	0.54
I71-84	14	65	691	0.57	0.48
I81-94	14	73	665	0.90	0.54
I71-91	21	100	973	0.77	0.48
I75-94	20	98	640	0.94	0.54
I72-92	21	102	1008	0.85	0.58
I74-94	21	102	1003	0.78	0.54
I75-95	21	103	993	0.85	0.54
I76-96	21	103	982	0.74	0.54
I78-98	21	106	975	0.97	0.54
I71-97	27	135	1284	0.61	0.54
I72-78	27	132	1287	0.58	0.58
I71-98	28	137	1311	0.63	0.54

Table 3: Experimental results on large instances.

Policella 2004; 2007) and in this paper. These versions of the problem, described for instance by (Cesta et al. 2007b; 2007a), can also be solved by mathematical programming, following the approach shown in this paper, even if the computational complexity can be much higher due to the possible presence of binary variables in the mathematical model. This is the subject of ongoing research.

Acknowledgments

We thank Alessandro Donati and Nicola Policella for their helpful collaboration. This work has been done while the second author was at Advanced Mission Concepts Technologies Office, ESA/ESOC, Darmstadt, with a trainee fellowship “Scheduler for Synthesizing Telecommands Uplink Activity”.

References

Ahuja, R.; Magnanti, T.; and Orlin, J. 1993. *Network flows*. Prentice Hall.

Cesta, A.; Oddi, A.; Cortellessa, G.; and Policella, N. 2002. Automating the generation of spacecraft downlink operations in mars express: analysis, algorithms and an interactive solution aid. Technical report, MEXAR-TR-02-10 (Project Final Report), ISTC-CNR, Italian National Research Council.

Cesta, A.; Cortellessa, G.; Fratini, S.; Oddi, A.; Denis, M.; Donati, A.; Policella, N.; Rabenau, E.; and Schulster, J. 2007a. Mexar2: A.i. solves mission planner problems. *IEEE Intelligent Systems* 22:12–19.

Cesta, A.; Cortellessa, G.; Fratini, S.; Oddi, A.; and Policella, N. 2007b. An innovative product for space mission planning: an a posteriori evaluation. In *Proc. of International Conference on Automated Planning and Scheduling, ICAPS-07*. AAAI.

Oddi, A., and Policella, N. 2004. A max-flow approach for improving robustness in a spacecraft downlink schedule. In *Proc. of the 4^{th} International Workshop on Planning and Scheduling for Space, IWPSS 2004*. ESA/ESOC, Darmstadt.

Oddi, A., and Policella, N. 2007. Improving robustness of spacecraft downlink schedules. *IEEE Transactions on Systems, Man and Cynernetics - part C: Applications and Reviews* 37:887–896.

Oddi, A.; Policella, N.; Cesta, A.; and Cortellessa, G. 2003. Generating high quality schedules for a spacecraft memory downlink problem. *Lecture Notes in Computer Science* 2833:570–584.