A Max-Flow Approach for Improving Robustness in a Spacecraft Downlink Schedule

Angelo Oddi and **Nicola Policella**
Planning & Scheduling Team [PST]
ISTC-CNR – Italian National Research Council
Viale Marx 15, I-00137 Rome, Italy
{a.oddi, policella}@istc.cnr.it

Abstract. In the realm of scheduling problems different sources of uncertainty can invalidate the solutions. In this paper we are concerned with the generation of high quality downlink schedules in a spacecraft domain in presence of a high degree of uncertainty. In particular, we refer to a combinatorial optimization problem called MARS EXPRESS Memory Dumping Problem (MEX-MDP), which arises in the European Space Agency program MARS EXPRESS. A MEX-MDP consists in the generation of dumping commands for transferring the whole set of data from the satellite to the ground. The domain is characterized by several kinds of constraints - such as, bounded on-board memory capacities, limited communication windows over the downlink channels, deadlines and ready times imposed by the principal investigators - and different sources of uncertainty - e.g., the amount of data generated at each scientific observation or the channel data rate. This work describes a reduction of the MEX-MDP to a Max-Flow problem, such that the problem has a solution when the maximum flow equates the total amount of data to dump. Based on this reduction, an iterative procedure is built to improve the robustness of a solution with respect to the utilization of the on-board memory. The underlying idea being that the lower are the peaks in memory utilization, the higher the ability of facing unexpectedly larger amount of data.

1 Introduction

In a space domain, as in many other applicative domains, the usefulness of schedules is limited by their brittleness. Though a schedule offers the potentials for optimized operations, it must in fact be executed exactly as planned to achieve this potential. In practice, this is generally made difficult in a dynamic execution environment, where unexpected events quickly invalidate the schedule's predictive assumptions and the validity of the schedule's prescribed actions is continuously brought into question. The lifetime of a schedule tends to be very short, and hence its optimizing advantages are generally not realized.

MARS-EXPRESS is an ESA program that has launched a spacecraft toward Mars on last June 2, 2003 and now, as it is well-known, the space probe is orbiting around the Red Planet and is operating seven different payloads. MARS-EXPRESS represents a challenging and interesting domain for research in automated problem solving. The domain is characterized by several kinds of constraints - such as bounded on-board memory capacities, limited communication windows over the downlink channels, deadlines and ready times imposed by the main investigators - and different sources of uncertainty - e.g., the amount of data generated at each scientific observation or the channel data rate.

This paper analyzes and models, through a Max-Flow paradigm, the so called MARS-EXPRESS Memory Dumping Problem (MEX-MDP). The problem involves the process of automating the memory dump operations of both science and housekeeping data, where we consider peaks of data in memory utilization as sources of brittleness in the schedule. As a consequence, we propose a novel iterative *leveling* algorithm, based on the Max-Flow reduction, to increase the robustness of a solution by finding a different distribution of the dumping operations over the same horizon through *flattening* the peaks in memory utilization. Problems similar to MEX-MDP can arise in satellite domains such as the ones described in (Verfaillie and Lemaitre 2001; Bensana *et al.* 1999). Both works concern a set of Earth observation operations to be allocated over time under a set of mandatory constraints such as: no overlapping images, sufficient transition times (or *setup* times), bounded instantaneous data flow and on-board limited recording capacity.

The paper is organized as follows. After a brief introduction of MEX-MDP, the novel flow network model is described. Then, a basic algorithm to solve a MEX-MDP instance and an iterative method to improve the schedule robustness are introduced. The paper ends with some experimental evaluations and a discussion about the future developments of the presented method.

2 The Memory Dumping Problem

In a deep-space mission like MARS-EXPRESS data transmission to Earth represents a fundamental issue. The space-probe continuously produces a large amount of data resulting from the activities of its payloads (e.g. on-board scientific programs) and from on-board device monitoring and verification tasks (the so called *housekeeping data*). All these data, usually referred to as *telemetry*, are to be transferred to Earth during downlink sessions. MARS-EXPRESS

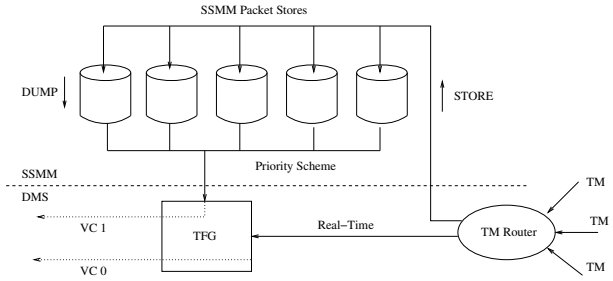


Figure 1: On-board telemetry flow

is endowed with a single pointing system, thus during regular operations, it will either point to Mars and perform payload operations or point to Earth and transmit data through the downlink channel. As a consequence on-board data are first stored in the Solid State Mass Memory (SSMM) and then transferred to Earth.

The main problem to be solved consists in synthesizing spacecraft operations for emptying as much as possible the on-board memory during the available downlink time intervals, in order to allow the spacecraft to save new information without losing previously stored data and to minimize a given objective function (for example, the average turnover time - or flow-time - for a set of scientific observations). Several of the constraints related to this problem are mutually conflicting. Besides the communication channel availability, different transmission rates are to be taken into account. Additional constraints arise from the specific use of the on-board memory. That memory is subdivided into different memory banks (or packet stores) each having a finite capacity. Also, for each piece of information produced inside the probe, one or more packet stores are identified in which such data should be stored. Different data are stored in a sequential way and the packet stores are managed cyclically. As a consequence, in case the memory is full, precious data might be overwritten as new data become available, and this is to be avoided as much as possible.

2.1 Basic Modeling

The Mars Express Memory Dumping Problem (MEX-MDP) has been formalized in a previous study conducted for the ESA (see (Cesta *et al.* 2002)). In the rest of this section we describe the main MEX-MDP components.

Figure 1 shows a sketch of the MARS EXPRESS modules that are relevant to MEX-MDP. It shows the different telemetry (TM) data produced on-board and then stored in the *Solid State Mass Memory (SSMM)* that is subdivided into packet stores. Memory stores are downloaded with different dumps that transfer data to Earth. The basic objects that are relevant to the MEX-MDP domain can be subdivided into either *resources* or *activities*.

Resources represent domain subsystems able to give services, while *activities* model tasks to be executed using resources over time. Our model of MARS-EXPRESS requires two types of resources:

- *Solid State Mass Memory (SSMM)*. The SSMM is able to store both science and housekeeping (HK) information. SSMM is subdivided into a set of *packets store* $\{pk_1, pk_2, \ldots, pk_m\}$, each one with a fixed capacity c_i and a priority p_i for dumping data. Each packet store can be seen as a file of a given maximal size and cyclically managed (previous information is overwritten if the amount of data overflow the packet store capacity).
- *Communication Channels*. The downlink connections to Earth for transmitting data. These resources are characterized by a set of separated communication windows $CW = \{w_i\}$ identifying intervals of time in which downlink connections can be established. Each element w_i is a 3-tuple $\langle r_i, s_i, e_i \rangle$, where r_i is the available data rate during the time window w_i and s_i and e_i are respectively the start and the end-time of such window.

Activities describe *how* resources can be used. Each activity a_i is characterized by a fixed duration d_i and two variables s_i and e_i which respectively represent its start-time and its end-time. Two basic types of activity are relevant to MEX-MDP: store operations st_i and memory dumps md_i.

- *Store Operation*. Each st_i "instantaneously" stores an amount of data q_i at its end-time in a destination packet store pk_i.
- *Memory Dump*. An activity that transmits a set of data from a packet store to the ground station.

In the MEX-MDP domain there are two different data sources requiring store operations st_i: the so-called *Payload Operation Request* (POR) and a set of housekeeping activities, which produce a continuous stream of data at a given constant rate, called *Continuous Data Stream* (CDS).

A *Payload Operation Request* is a model for a scientific observation which generates a set of data distributed over a subset of the available packet stores. According to this model, the produced data are decomposed in a set of different *store* operations such that, $por_i = \{st_{ij}\}$, all of them with the same durations and start-times.

On the other hand, a *Continuous Data Stream* models an on-board process which works in "background" with respect to the scientific activities. It generates a flow of data with constant rate which has to be stored in the SSMM. Examples of such data streams are the housekeeping data collected on a regular basis to control the behavior of the on-board subsystems.

The two data sources exhibit different characteristics: in fact, a POR is a time bounded activity, which stores data at its end-time, whereas a CDS is a continuous data flow over the domain horizon. However, we choose to model

also a CDS as a periodic sequence of store operations. In particular, given a CDS with a flat rate r, we define a period T_{cds}, such that, for each instant of time $t_j = j \cdot T_{cds}$ ($j = 0, 1, 2, \dots$) an activity st_{ij} stores an amount of data equal to $r \cdot T_{cds}$. In the particular case of $t_0 = 0$ we suppose the amount of stored date is zero. Hence, we can consider as input data for the problem just an equivalent set of store operations containing data packets, such that, each packet contains a pointer to its source.

2.2 Problem Definition

Given these basic domain entities, let us now define the MEX-MDP. A set of scientific observations, $\mathcal{POR} = \{por_1, por_2, \dots, por_n\}$ and a set of housekeeping productions, $\mathcal{CDS} = \{cds_1, cds_2, \dots, cds_m\}$, are both reduced to a set of store operations on the on-board memory. A *solution* to a MEX-MDP, is a set of dumping operations $S = \{md_1, md_2, \dots, md_s\}$ such that:

- the whole set of data are "available" on ground within the considered temporal horizon $\mathcal{H} = [0, H]$.
- Each dump operation starts after the generation of the corresponding data. For each packet store, the data are moved through the communication channel according to a First In First Out (FIFO) policy.
- Each dump activity, md_i, is executed within an assigned time window w_j which has a constant data rate r_j. Moreover, dump operations cannot mutually overlap.
- At each instant $t \in \mathcal{H}$, the amount of data stored in each packet store pks_i has to be less or equal to the packet store capacity c_i (i.e., overwriting is not allowed).

Though a solution should satisfy all the imposed constraints, however our main goal is to find *high quality* solutions with respect its *robustness*. Informally, *a high quality plan delivers all the stored data and is able to adsorb external modifications that might arise in a dynamic execution environment*.

Our definition of solution's robustness is related to the idea of *distance of the solution to the overwriting state*. In other words, we consider peaks of data in a packet store close to its maximal capacity as sources of schedule's brittleness and a way to increase robustness is to *flat* these peaks by finding a different distribution of the dumping operation over the same problem horizon. In particular, given a solution to a MEX-MDP instance, for each packet store pk_i it is possible to give a time function $use_i(t)$, with $t \in [0, H]$, representing the amount of data memorized in the packet store pk_i at the instant t. Let $use_i^{(max)}$ the maximal value over the horizon $[0, H]$ of $use_i(t)$. We define the packet store utilization $\alpha_i = use_i^{(max)}/c_i$ as the ration between the maximal level of data in the packet store pk_i and its capacity c_i. The robustness of a solution S is defined as:

$$r(S) = max_{i=1..m}\{\alpha_i\} \quad (1)$$

that is the maximal value of utilization over the set of packet stores. A solution S is *optimal* when $r(S)$ is minimal.

3 A Max-Flow Approach for MEX-MDP

A MEX-MDP instance concerns the synthesis of a set of dump commands for transferring to the ground the whole set of data collected by the satellite. To face such a problem the following abstraction has been adopted: on a first level, called *Data Dump level*, it is assessed the amount of data to dump from each packet store for each time window. In a successive level, called *Packet level*, the final dump commands are generated from the first level results. This second step can be done automatically once a solution for the Data Dump level is achieved. For this reason in this work we focus our attention exclusively on the Data Dump level. This problem *decomposition* is motivated by the complexity of the optimization problem. Using this abstraction we focus on the *dominant* aspects of the problem that consist of reasoning on data quantities, packets store capacities and dump capability over the communication links, without considering the problem of decomposing the dumped data into dump commands.

In the remainder of the section we introduce a formalization for MEX-MDP as a Max-Flow problem and the flow network associated to a MEX-MDP instance.

3.1 A formalization for MEX-MDP

The formalization is based on a partition of the temporal horizon $\mathcal{H} = [0, H]$ into a set of contiguous windows $W = \{w_1 = [t_0, t_1] \mid t_0 = 0\} \cup \{w_j = (t_{j-1}, t_j] \mid j = 2 \dots m, t_i \in \mathcal{H}\}$, such that $\cup_{i=1}^{m} w_i = \mathcal{H}$. The partition is realized upon consideration of significant events. Such events are assumed to be the start and the end of the temporal horizon, the time instants where a memory reservation on a packet store is performed, and the time instants where a change on the channel data rate is operated. It is assumed that such significant events take place at the windows' edges.

The key point of the formalization is represented by the variables:

$$\delta_{ij} \qquad i = 1..n,\ j = 1..m, \quad (2)$$

each defined in the domain $[0, \infty)$. These represent the amount of data to dump from the packet store pk_i within a window w_j. To formally represent the domain constraints, for each packet store pk_i ($i = 1..n$) and for each time window w_j ($j = 1..m$) some additional definitions are needed:

- d_{ij}, amount of data memorized in the packet store pk_i at t_j. Where the variables $d_{i0} \leq c_i$ represent the initial data level in the packet store pks_i;
- l_{ij}, maximal level (amount of data stored) allowed at t_j for the packet store pk_i, $l_{ij} \in [0, c_i]$;
- b_j, maximal dumping capacity available in w_j;

We introduce two classes of constraints on the set of decision variables δ_{ij}. A first constraint captures the fact that for each window w_j the difference between the amount of generated data and the amount of dumped data cannot exceed the maximal imposed level in the window l_{ij} (*overwriting*). Additionally, the dumped data cannot exceed the generated data (*overdumping*). We define the following inequalities as *conservative constraints*:

$$\sum_{k=0}^{j} d_{ik} - \sum_{k=1}^{j} \delta_{ik} \leq l_{ij} \quad (3)$$

$$\sum_{k=0}^{j-1} d_{ik} - \sum_{k=1}^{j} \delta_{ik} \geq 0 \quad (4)$$

for $i = 1 \ldots n$ and $j = 1 \ldots m$. The store operations d_{ij} are 'impulsive actions' performed at the end of each time window w_j, whereas the dumping operations δ_{ij} are performed during the window w_j. As a consequence, the amount of data d_{ij} cannot be dumped in the window w_j, because the data are not available during w_j. Hence, we have to check the constraint (4) to avoid dumping more data than the available amount at $t_{i(j-1)}$ and the constraint (3) at t_j, to avoid storing more data than the allowed level l_{ij}.

A second class of constraints concerns the dumping capacity imposed by the communication channel:

$$0 \leq \sum_{i=1}^{n} \delta_{ij} \leq b_j \quad j = 1..m \quad (5)$$

these inequalities, called *downlink constraints*, state that for each window w_j is not possible to dump more than a certain amount of data (i.e. b_j).

3.2 The Max-Flow Problem

In the following we briefly review the theory behind the Max-Flow problem (Cormen *et al.* 2001). A flow network $G(V, E)$ is a direct graph where V is a set of vertices and E is a set of edges (u, v) with nonnegative capacity $c(u, v) \geq 0$. The flow network has two special vertices: a source s and sink t. A flow in G is a real-valued function $f : V \times V \rightarrow \Re$ that satisfies the following three properties:

- for all $u, v \in V$, $f(u, v) \leq c(u, v)$
- for all $u, v \in V$, $f(u, v) = -f(v, u)$
- for all $u \in V \cup \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$.

the value of a flow f into the graph G, is defined as

$$f = \sum_{v \in V} f(s, v),$$

that is the total flow out of the source. In the Max-Flow problem given a flow network G, the goal is to find a flow of maximum value from source to sink.

3.3 The Max-Flow Model for MEX-MDP

The current paragraph aims at introducing the flow network model associated to a MEX-MDP instance. Figure 2 shows an example of flow network in the case of two packets store, however this example can be easily generalized to the case of n packets stores. There are four types of nodes: *source*, *sink*, packet-store nodes p_{ij} (as explained below such nodes are actually *macro-nodes*) and channel nodes ch_j.

The packet store node p_{ij} is composed of two nodes to represent the two conservative constraints (see Fig. 3). The first node $p_{ij}^{(1)}$ represents the *overdumping* constraint (4), such that, within the window w_j it is not possible to dump more data than the amount available at t_{j-1}. On the other hand, the node $p_{ij}^{(2)}$, represents the *overwriting* constraint (3), such that, the amount of residual data in the packet store after the dumping operation over the window w_j, added to the amount of data stored at t_j (d_{ij}), is less or equal to the allowed capacity of the packet store l_{ij}. In particular, on the node $p_{ij}^{(1)}$ there are the following three flows: $f(p_{i(j-1)}, p_{ij})$, $f(p_{ij}, ch_j)$, and $f(p_{ij}^{(1)}, p_{ij}^{(2)})$. The flow $f(p_{i(j-1)}, p_{ij})$ represents the residual amount of data on pk_i at t_{j-1}, that is, $\sum_{k=0}^{j-1} d_{ik} - \sum_{k=1}^{j-1} \delta_{ik}$.The value $f(p_{ij}, ch_j)$ represents the amount of data dumped from pk_i during the time window w_j, that is $\delta_{ij} = f(p_{ij}, ch_j)$. This edge is labeled with $c(p_{ij}, ch_j) = b_j$ to hold the constraint $\delta_{ij} \leq b_j$, that is, *it is not possible to dump more data than the channel capacity* b_j. Finally the flow $f(p_{ij}^{(1)}, p_{ij}^{(2)})$ represents the amount of residual data which remains in the packet store after the dumping over the window w_j.

We remark that we split the node p_{ij} for the same motivation above explained for the inequalities (3) and (4). In fact, if we consider the flow balance on the node $p_{ij}^{(1)}$ we have that:

$$f(p_{i(j-1)}, p_{ij}) - f(p_{ij}, ch_j) = f(p_{ij}^{(1)}, p_{ij}^{(2)})$$

that can be rewritten as:

$$\sum_{k=0}^{j-1} d_{ik} - (\sum_{k=1}^{j-1} \delta_{ik} + \delta_{ij}) = f(p_{ij}^{(1)}, p_{ij}^{(2)}) \geq 0$$

which coincides with the inequality (4). On the other hand, on the node $p_{ij}^{(2)}$ there are the following flows: $f(p_{ij}^{(1)}, p_{ij}^{(2)})$, $f(s, p_{ij})$, and $f(p_{ij}, p_{i(j+1)})$, where the latter represents the residual amount of data $\sum_{k=0}^{j} d_{ik} - \sum_{k=1}^{j} \delta_{ik}$, and the flow $f(s, p_{ij})$ concerns the amount of data stored at the window w_j, which can be at most equals to d_{ij}. Considering the flow balance on $p_{ij}^{(2)}$ we have that:

$$f(p_{ij}^{(1)}, p_{ij}^{(2)}) + f(s, p_{ij}) = f(p_{ij}, p_{i(j+1)})$$

that can be rewritten as:

$$(\sum_{k=0}^{j-1} ik + d_{ij}) - \sum_{k=1}^{j} \delta_{ik} = f(p_{ij}, p_{i(j+1)}) \leq l_{ij}$$

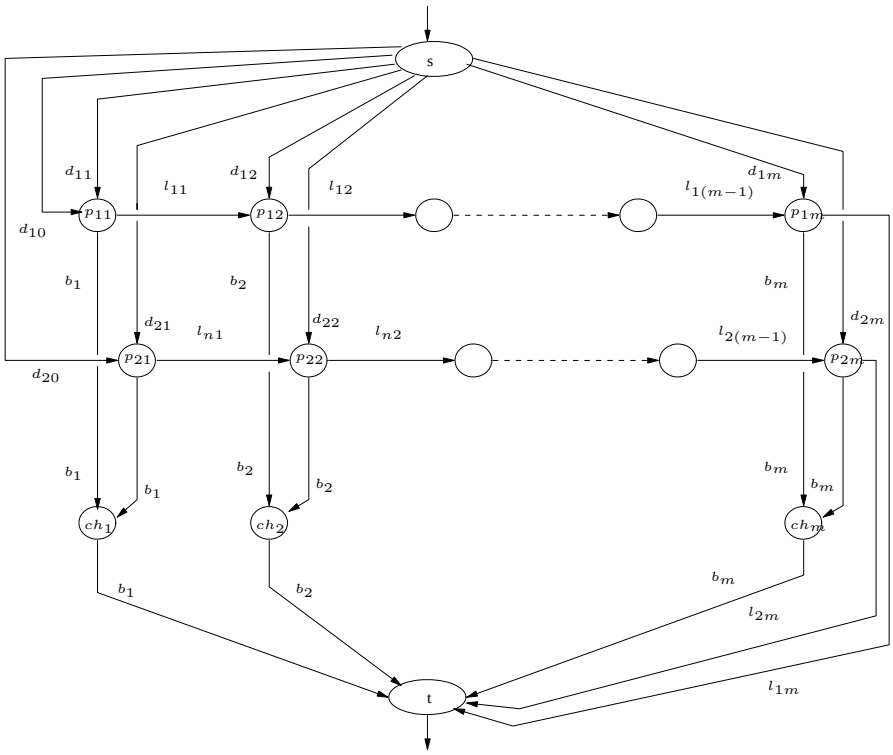


Figure 2: A flow network for MEX-MDP.

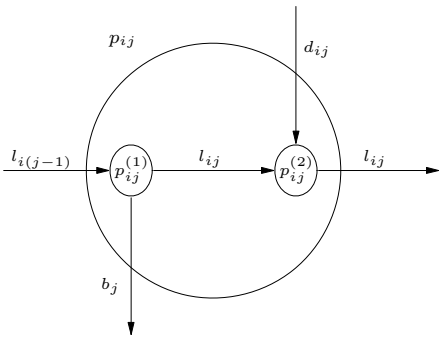


Figure 3: The internal structure of a macro-node p_{ij}.

which coincides with the inequality (3).

To represent the initial value of each packet store the arcs $(s, p_{i0}^{(1)})$ are labeled with the value d_{i0} whilst the arcs $(p_{im}^{(2)}, t)$ are used to represent the overall residual amount of data on the packets store pk_i. To synthesize solutions which dump all the data on the ground these arcs are labeled with $l_{im} = 0$.

Finally the flow through each channel node ch_j represents the *downlink* constraint over the time window w_j,

$$\sum_{i=1,\ldots,m} f(p_{ij}, ch_j) = \sum_{i=1,\ldots,m} \delta_{ij} = f(ch_j, t) \leq b_j$$

that is, *the sum of data dumped from each packet store over the window w_j is less or equal the channel capacity b_j*.

To conclude, a flow assignment to any edges is computed by applying a Max Flow algorithm to the flow network in Fig. 3. The result represents the maximum value of data which can be conveyed through the communication channel. Thus a MEX-MDP will admit a solution **if and only if**:

$$f(s, p_{ij}) = c(s, p_{ij}) = d_{ij}$$

for each $i = 1, \ldots, n$ and $j = 1, \ldots, m$. In other words, the formula above states that the maximum flow through the network equates the whole data set stored in the different packets store.

Based on the model described above, in the next section two methods for approaching a MEX-MDP instance are introduced: the former computes a solution, while the latter aims at refining an initial solution for the sake of robustness.

4 Solving Methods

As mentioned above, to find a solution to the MEX-MDP problem is sufficient to apply a Max-Flow algorithm to the associated flow network (see Figure 3). A solution of a problem instance mdp exists if and only if the flow through each

```
ITERATIVE-LEVELING(mdp, ε)
 1  δ ← SOLVEBYMAXFLOW(mdp)
 2  if δ ≠ ∅ then
 3        for j = 1 to m
 4              flatten[k] ← TRUE
 5        while ∃i|flatten[i] =TRUE
 6              pk_k ← SELECTPACKETSTORE()
 7              for j = 1 to m
 8                    l_kj ← α_k(1 − ε)l_kj
 9              δ' ← SOLVEBYMAXFLOW(mdp)
10              if δ' = ∅ then
11                    flatten[k] ← FALSE
12                    for j = 1 to m
13                          l_kj ← l_kj / (α_k(1−ε))
14                else δ ← δ'
15  return δ
```

Figure 4: Iterative-Leveling algorithm.

arc (s, p_{ij}) equates the capacity value, that is, $f(s, p_{ij}) = c(s, p_{ij})$. In such a case we have that the set of values $\delta_{ij} = f(p_{ij}, ch_j)$ is a solution of mdp. There are different polynomial algorithms to solve the Max-Flow problem. Our current implementation is based on the Edmond-Karp version of the Ford-Fulkerson method. The Edmond-Karp algorithm runs in $O(|V||E|^2)$ time, where $|V|$ and $|E|$ are respectively the number of nodes and the number of arcs in the flow network. Being in our case $|V| = O(m\,n)$ and $|E| = O(m\,n)$, our current implementation runs in $O(m^3n^3)$. An improvement in the sense of CPU time can be achieved using more sophisticated max flow methods like the preflow-flush one (Goldberg and Tarjan 1988). The reader can find an essential survey on this issue in (Cormen *et al.* 2001, chapter 26).

4.1 Iterative leveling

In this section we present an iterative algorithm for improving the robustness of an input solution. We recall that in this domain we consider a solution robust if the level of data over time of each packet store has no peaks close to its maximal capacity, such that there is always available memory for unexpected larger amount of data.

We propose an heuristic algorithm for improving robustness which iteratively applies the three-steps cycle *solution analysis/problem update/construction*. Our approach somehow resembles the concept of "feedback" widely used in Control Theory. Furthermore, a similar issue has been also used in the work (Joslin and Clements 1999) for the optimization of the makespan of scheduling problems.

Figure 4 shows the algorithm. It takes in input a MEX-MDP instance mdp, and a parameter $\epsilon \in (0, 1)$. The algorithm starts by finding an initial solution solution, represented in compact way by $\delta = \{\delta_{ij}\}$ (line 1). If a solution is found, $\delta \neq \emptyset$, the algorithm proceeds initializing all the elements of the vector $flatten[]$ at TRUE, where $flatten[i] =$ TRUE means that the usage of the packet store pk_i can be improved.

The **while** loop (Steps 5-14) represents the core of the algorithm. In this the following three steps are iteratively repeated:

1. analyze the current solution and selects a *critical* packet store pk_k, such that the percentage usage value α_k is maximum (Step 6);
2. for any time window w_j, the capacity l_{kj} of the selected packet store pk_k is reduced to the value $\alpha_k(1 - \epsilon)l_{kj}$ (Steps 7-8). In this way the maximum percentage usage is forced to be less than α_k;
3. solve the modified problem (Step 9). If this does not admit a solution then the "modified" packet store pk_i, is labeled as not improvable $flatten[i] =$ FALSE and the previous consistent situation is reloaded (Steps 11-13). Otherwise the current best solution is updated (Step 14).

The aim of the three steps is to iteratively flat the current critical packet store. These steps will be repeated until there is at least one packet store which admit an improvement.

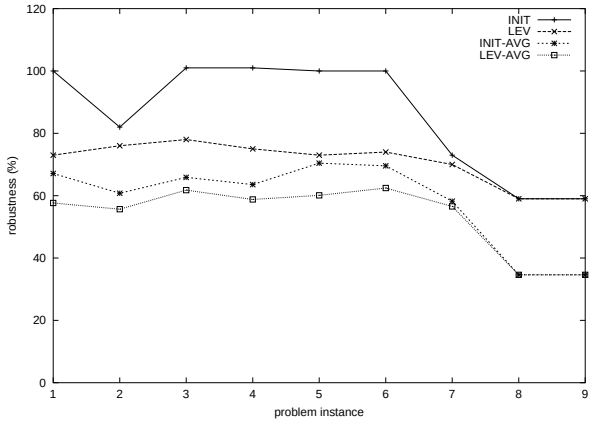
5 Experimental Evaluation

The method described above has been evaluated using the benchmark sets defined during a study conducted for the ESA and described in (Cesta *et al.* 2002). In this section we present the results for one of these benchmark sets[1], B5. This benchmark is composed of 9 problems and is one of the most critical with respect to, on one hand, the competition among the packet stores for the same channel bandwidth, and, on the other hand with respect to the limited capacity of the packet stores relatively to the amount of generated data on the other.

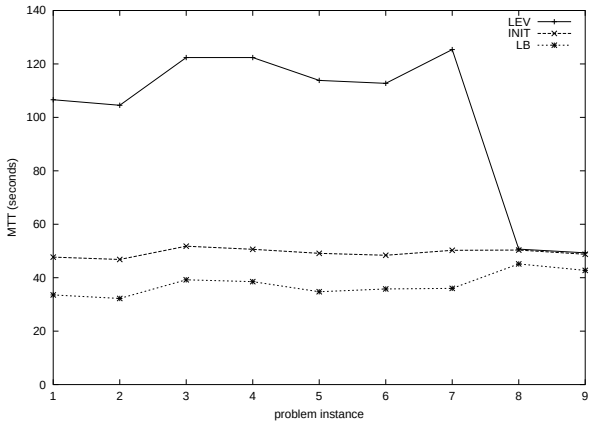
All the algorithms presented in this paper are implemented in Java on an Athlon 1800 Mhz machine. Figure 5(a) shows how the application of the *Iterative-Leveling* algorithm can improve the robustness (see Section 2) of a solution. In particular, the graph labeled with $INIT$ represents the robustness of a solution generated with one run of the solving algorithm based on the Max-Flow reduction. Whereas the graph labeled with *LEV* represents the robustness values after the application of the *Iterative-Leveling* algorithm described in Section 4 with $\epsilon = 0.02$. As Figure 5(a) shows, for some problems the robustness is improved from 100% to 75%. The average CPU-times are respectively 0.8 seconds to generate an initial solution and 21.8 seconds to improve its robustness.

In addition, two further curves, *INIT-AVG* and *LEV-AVG*, are shown in Figure 5(a). These represent the average of the

[1]The benchmark sets are available at the address: `http://mexar.istc.cnr.it`.



(a) Robustness values



(b) MTT values

Figure 5: Performance on benchmark B5

set of values $use_i^{(max)}$ over the set of the used packet stores respectively before and after the application of the *Iterative-Leveling* algorithm. Clearly, the main effect of the leveling algorithm is to create a different distribution of the dumping operations over the horizon in order to remove *dangerous* data peaks.

One more aspect we want to highlight is the following: the new distribution of data over the problem horizon comes at a price in term of other performance measures. In a previous work we have described several heuristic strategies to solve MEX-MDP as a combinatorial optimization problem to minimize the so called *mean turnover time* (MTT) (Oddi *et al.* 2003). The turnover time of a single payload operation por_i coincides with the time elapsed from the store of the data in the Solid State Mass Memory device, SSMM, to its delivery to Earth. In Figure 5(b) we present MTT values for the same set of problems used for robustness evaluation. In particular, the curves labeled with *INIT* and *LEV* represent the MTT values obtained through the application of the iterative sampling optimization strategy described in (Oddi *et al.* 2003) respectively with and without the application of the leveling algorithm described above. A quality decrease in term of MTT is evident when the robustness performance is increased. In fact, we observe that in order to improve the mean turnover time, a simple and effective heuristic is to dump first the activities with the smallest amount of data. On the other hand, the less are the capacities of the packets store, the fewer are the chances to follow the above heuristic, because activities involving larger amount of data should be dumped before less demanding activities in order to satisfy the memory capacity constraints. In general, it is a well-known concept that the improvement of an objective measure often stems from the decay of another one. However, in our case the price to pay is not so high, in fact the obtained MTT values when the the objective function is scheduling robustness are only twice the best MTT value obtained. Since the average values are around one minute, the new values are clearly acceptable.

6 Future Work

Generating high quality schedules for spacecraft downlink scheduling problems can hardly be seen as a single objective optimization problem, but rather as an optimization problem involving multiple, conflicting and non-commensurate criteria. MEXAR project (Cortellessa *et al.* 2004) has started a research path along in this direction with the main goal of defining a Decision Support System (DSS) for solving MEX-MDP [2]. Within this project the main idea is to integrate human strategic capabilities and automatic problem solving algorithms to find solutions with *the right compromise among different and contrasting goals*. Our future work will be still focused along this path, in particular, a remarkable aspect to pursue will be the integration of the method presented in this paper with an interaction module. Such a module should provide the user with the ability to analyze the solution and, more importantly, to participate to the leveling process.

7 Conclusions

This paper has introduced a novel approach to solve the so called MARS-EXPRESS Memory Dumping Problem (MEX-MDP). A problem arisen during a project work for the European Space Agency (Cesta *et al.* 2002). The work describes a reduction of the MEX-MDP to a classical problem: the Max Flow problem (Cormen *et al.* 2001). The reduction can be intuitive considering that a solution to the dumping problem can be seen as a flow from the satellite to the ground, such that the problem has a solution when the maximum flow equates the total amount of data to dump.

Given this reduction, a novel definition of solution robustness is proposed together an iterative procedure to improve

[2] For further information see the MEXAR project home page at `http://mexar.istc.cnr.it`.

the robustness of a solution. The underlying idea being that the lower the memory utilization the higher the ability of facing unexpectedly larger amount of data.

Some experimental data support our thesis: we can effectively remove *dangerous* peaks of data and distribute them over the problem horizon by using the *Iterative-Leveling* procedure. However, as it is expected, this process comes at a price as other solution quality measures, for example the mean turnover time, might get worse. This opens interesting research scenarios toward the definition of more effective procedures for Decision Support Systems and Mixed-Initiative problem solving approaches, where we we have to consider the problem from a multi-objective optimization point of view.

We remark that even though the MEX-MDP problem comes from a specific study, its features are quite general and many of the conclusions reported in this paper can be extended to other spacecraft domains which adopt the same model of on-board memory. In particular, the Max-Flow reduction allows to check the existence of a solution in polynomial time, and this property might be used in many different contexts, as the definition of a multi-objective optimization procedure or a Decision Support System. In this work we have used this reduction to propose a fast heuristic approach to improve robustness in a solution.

Acknowledgements. The MEX-MDP has been studied in a study conducted for ESA from November 2000 to July 2002 (contract No. 14709/00/D/IM). The authors are currently supported by ASI (Italian Space Agency) under projects SACSO and ARISCOM. The authors would like to thank their colleague Riccardo Rasconi for his precious suggestions.

References

[Bensana *et al.* 1999] E. Bensana, M. Lemaitre, and G. Verfaillie. Earth Observation Satellite Management. *Constraints: An International Journal*, 4(3):293–299, 1999.

[Cesta *et al.* 2002] A. Cesta, A. Oddi, G. Cortellessa, and N. Policella. 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 [PST], Italian National Research Council, July 2002.

[Cormen *et al.* 2001] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, Second Edition. MIT Press, 2001.

[Cortellessa *et al.* 2004] G. Cortellessa, A. Cesta, A. Oddi, and N. Policella. User Interaction with an Automated Solver. The Case of a Mission Planner. *PsychNology Journal*, 2(1):140–162, 2004.

[Goldberg and Tarjan 1988] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of ACM*, 35(4):921–940, October 1988.

[Joslin and Clements 1999] D.E. Joslin and D.P. Clements. "Squeaky Wheel" Optimization. *Journal of Artificial Intelligence Research*, 10:353–373, 1999.

[Oddi *et al.* 2003] A. Oddi, N. Policella, A. Cesta, and G. Cortellessa. Generating High Quality Schedules for a Spacecraft Memory Downlink Problem. In F. Rossi, editor, *Principles and Practice of Constraint Programming, 9^{th} International Conference, CP 2003*, number 2833 in Lecture Notes in Computer Science, pages 570–584, Kinsale, Ireland, 29 September - 3 October 2003. Springer.

[Verfaillie and Lemaitre 2001] G. Verfaillie and M. Lemaitre. Selecting and Scheduling Observations for Agile Satellites: Some Lessons from the Constraint Reasoning Community Point of View. In T. Walsh, editor, *Principles and Practice of Constraint Programming, 7^{th} International Conference, CP 2001*, number 2239 in Lecture Notes in Computer Science, pages 670–684. Springer, 2001.