

OPTIMALLY SOLVING NADIR OBSERVATION SCHEDULING PROBLEMS

Russell Knight and Ben Smith
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109-8099
{firstname.lastname}@jpl.nasa.gov

Abstract: We present optimal algorithms for nadir (instrument pointing straight down) observation scheduling for spacecraft with fixed orbits. We present a comparison between an integer program formulation and a branch and bound formulation that makes use of a flow network heuristic, each capable of solving instances of these problems optimally. Neither technique strictly dominates the other, and we characterize their respective advantages and disadvantages. Note that this problem is NP-complete..

1. INTRODUCTION

Nadir is the opposite of zenith; it is basically “straight down.” Orbiting spacecraft often have immobile imaging instruments, and generally such spacecraft maintain a fixed orientation with respect to the body that they are orbiting, therefore most instruments point straight down, or nearly straight down. This is called either nadir or off-nadir observing. Figure 1 (right) shows a nadir pointing spacecraft orbiting a spherical body and the area that is in view of the instrument. (The arrow indicates the direction of travel.)

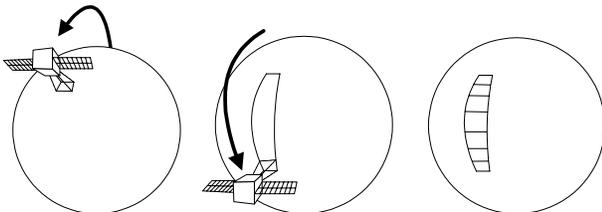


Figure 1 Swath and segments of a nadir pointing spacecraft

In these cases, the instrument gathers data along a fixed trajectory called a swath. Figure 1 (center) shows the swath that could be imaged by the orbiting spacecraft. In practice, the swaths are broken into smaller swaths called segments. A segment is the smallest area that can be individually imaged. Therefore, a single observation equates to a segment. It is important to note that segments are both areas that could be imaged and intervals of time for the observation. Figure 1 (left) shows some example segments.

The purpose of such spacecraft is to image various areas, called *targets*, according to the investigator’s priorities. Figure 2 shows an example of a target.

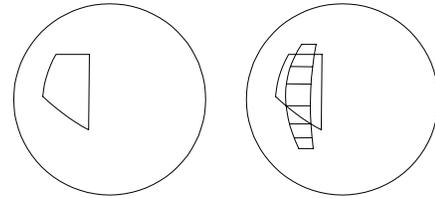


Figure 2 A target, without and with segments
Overlapping segments imply a potential for waste if the overlapping area is collected and transmitted more than once. Figure 3 shows more segments as a result of a subsequent orbit by the spacecraft. (The arrow indicates the direction of travel.) Note that the segment in the center of the target (shaded area) overlaps one of the previous segments.

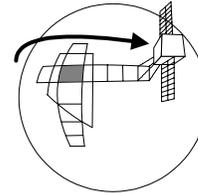


Figure 3 Another swath and its associated segments, with an overlapping segment (shaded area)
Collecting overlapping segments is a problem because there are limits to how many segments we may collect. This is usually due to limited on-board memory, and limited downlink times and capacities. Not surprisingly, segments are usually carefully chosen to reduce overlap. But choosing the best segments can be problematic, especially for large numbers of observations. This problem is called the *swath segment selection problem*. Note that the terms *swath* and *segment* are standard terms.

1 Problem Description

The Swath Segment Selection Problem (SSSP) consists of selecting a subset of data collection opportunities from all that are available such that the most valuable data are collected given the limitations of bounded memory and bounded communication. In practice, all possible segments (candidate observation times) are usually determined beforehand. This problem is NP-complete (see Appendix A).

1.1 Definitions

The swath segment selection problem (SSSP) consists of: a set of target polygons, a set of swath segments, a set of downlinks, and a memory capacity. From the segments, choose a subset that respects the memory capacity and downlink capacity that maximizes the area of the targets downlinked.

A *shard* is a sub-section of a target. We use shards to represent pieces of the target that can be gathered and downlinked. They are the natural result of combining the target and the edges of the segments. The term *shard* is taken from the basic appearance of these polygons as shards of broken glass, especially in larger problem instances. Figure 4 (left) shows a set of example shards derived from the segments and the target. We draw dotted lines around the inside of each shard for easier identification. For our formulation, we will assume that the targets are already broken into shards, and will therefore only refer to the shards. Note that the term *shard* is not standard and used solely for our description.

Thus, more formally, given:

- a set of shard polygons H where each $h \in H$ is a simple (but possibly concave) polygon in the Euclidean plane and the real-valued reward function $reward(h)$ that represents the reward for collecting the shard (piece of a target), e.g. $\{\alpha, \beta, \chi, \delta, \varepsilon, \phi\}$ (see Figure 4, left),
- a set of swath-segments S where each $s \in S$ is a convex quadrilateral in the Euclidean plane (these are not necessarily parallel due to viewing angle warping the projection of the instrument on the surface to be imaged), e.g., $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,\}$ (see Figure 4, right), the real-valued capacity cost function $cap(s)$ that represents the memory required to store the segment, and the function $shards(s)$ that returns the set of shards that intersect with s .
- a set of downlinks D where each $d \in D$ has a real-valued capacity function $cap(d)$ that represents the maximum amount of memory that can be communicated during the downlink, e.g., $\{D_1, D_2\}$.
- a memory limit m that represents the maximum amount memory that can be stored between downlink operations.
- a route \mathbf{R} that is a permutation of the segments and downlinks, e.g., $1, 2, 3, 4, 5, 6, D_1, 7, 8, 9, 10, 11, D_2$.

Find a solution route \mathbf{R}' that is a subset of \mathbf{R} that maximizes the reward of the *collected shards* while respecting memory capacity limits.

The collected shards are the union of $shards(s \in \mathbf{R}')$. Respecting memory capacity limits means that the sum of all segments previous to a given $d \in D$ but not previous to any other $d \in D$ must be less than m and less than $cap(d)$.

We presume real-valued functions $area(s \in S)$ and $area(h \in H)$ that gives the area of any segment or shard.

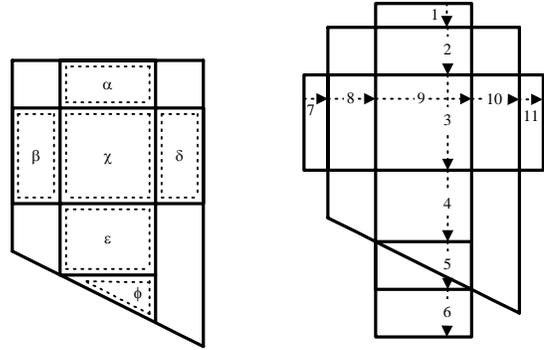


Figure 4 Example shards and segments

We continue with two solution examples. Let us consider that there is a downlink opportunity after between segments 6 and 7 that can transfer up to 32 units of memory, and a downlink opportunity after segment 11 that can transfer the same amount of memory. The total memory on board is 33 units. Table 1 shows us the area of each segment. One solution \mathbf{R}' is 2, 3, 5, D_1 , 8, 10, D_2 (as shown in the left of Figure 5). This respects the downlink limits as segments 2, 3, and 5 (32 total units) can be handled by the first downlink, and segments 8 and 10 (16 total units) can be handled by the second downlink. The collected shards are $\alpha, \beta, \chi, \delta,$ and ϕ . The quality of this solution is the collected target area (the summed area of the collected shards) – 44 units. The optimal solution, however, is 2, 4, 5, D_1 , 8, 9, 10, D_2 (as shown in the right of Figure 5). 2, 4, and 5 (28 total units) handled by the first downlink, and 8, 9, and 10 (32 total units) handled by the second. This results in all shards being collected, yet still respects the downlink limits, for a quality of 56 units.

| segment | area | shard | area |
|---------|------|---------------|------|
| 1 | 4 | α | 8 |
| 2 | 8 | β | 8 |
| 3 | 16 | χ | 16 |
| 4 | 12 | δ | 8 |
| 5 | 8 | ε | 12 |
| 6 | 8 | ϕ | 4 |
| 7 | 4 | | |
| 8 | 8 | | |
| 9 | 16 | | |
| 10 | 8 | | |
| 11 | 4 | | |

Table 1 Example segment and shard areas

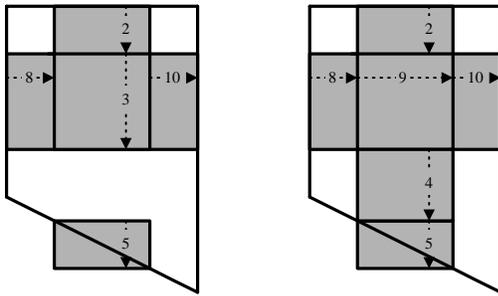


Figure 5 Example solutions

2 Solution Techniques

The state of the art system for solving the SSSP is the ASTER scheduling algorithm [Muraoka 1998]. This algorithm partitions the problem into a single day's worth of observations, and then solves each day's observations in order using a greedy technique (greatest reward), breaking ties by choosing the earliest segment. There may be many months in a given problem. This is not an optimal technique, but it is very fast and allows the users to try various schedules very quickly. This technique would give the solution $\mathbf{R}' = 2, 3, 5, D_1, 8, 10, D_2$, as is illustrated by Figure 5.

Our goal, however, is to solve problems as optimally as possible. In fact, our solutions are the first optimal solutions for these problems. We compare our approach to a straightforward straw-man integer program formulation (IP) and to the ASTER algorithm.

2.1 Solution Approach

Our approach searches the space of segment-inclusions, starting with a solution that includes no segments. We use a depth first branch and bound search with a heuristic award estimator. The heuristic award estimator is a network flow formulation of the problem. The node ordering heuristic orders selections based on the reward to capacity cost ratio of a segment, given the segments that are known to be included or excluded. We refer to this technique as *Flow*.

2.2 Network Flow Formulation

The goal is to generate a flow network that represents the flow of capacity usage through the problem. It is important to note that this formulation does have some limitations; most importantly it assumes that the reward for an element scales with its capacity cost, which might not be the case. Under those circumstances, the IP formulation is the more accurate, and probably should be used. The rest of this subsection describes the construction of the flow network. Given our previous example, we would first add a node called *src* and a node called *snk* to our graph (see Figure 6.)



Figure 6 Source and sink nodes for the flow network. We then add a node for each h in H , and add an edge representing the reward for collecting the shard from *src* to the elements node (see Figure 7). Note that capacity cost and reward must be equivalent for the network flow formulation to be used.

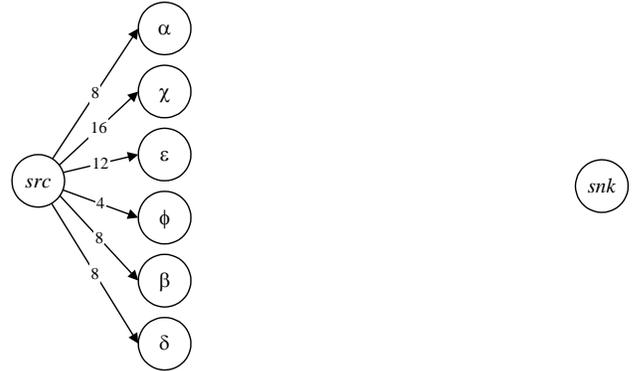


Figure 7 Shard nodes and reward edges

Then, for each segment, we add a node and add an edge from each shard that the segment contains with the same capacity as the reward for the shard (see Figure 8). Note that in our example, shard χ belongs to segment 3 and segment 9.

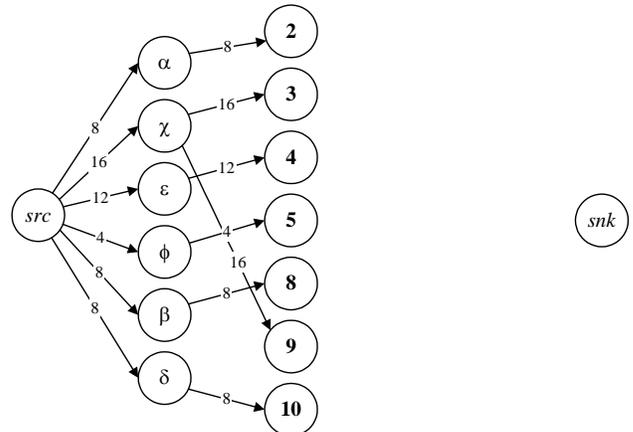


Figure 8 Segment nodes and shard-intersection edges

Then, for each downlink, we add two nodes. One node collects all of the segments, and we designate it the *in* node. The other sends the collected reward to the sink. So, for all segments of the leg previous to the downlink, we add an edge of the same capacity as the segment from each segment to the in node. We then add an edge of the same capacity as m between the in node and the second node. Finally, we add an edge of the downlink capacity from the second node to *snk*. Figure 9 shows the final flow network for our example problem, where the edge-labels represent the capacity of the edge. Figure 10 shows one possible solution to the network flow problem, where the edge-

labels represent the flow, and Figure 11 shows what this solution would imply when translated back into a swath segment selection problem.

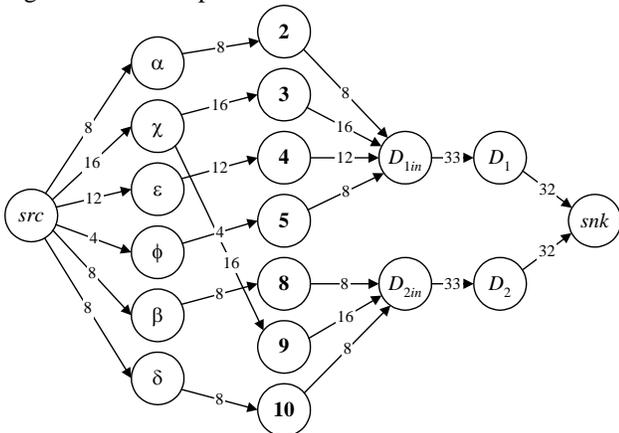


Figure 9 Flow Network example

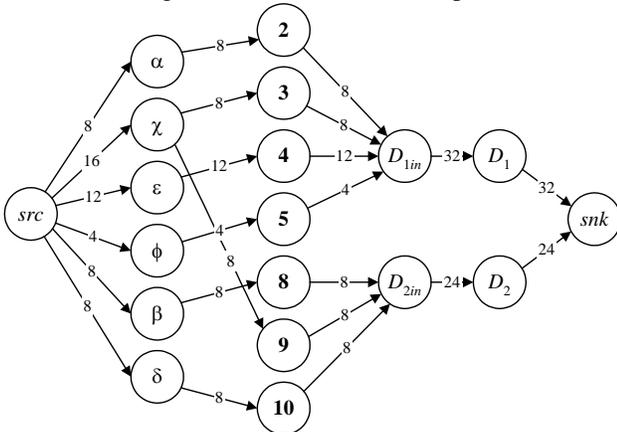


Figure 10 Example network flow solution

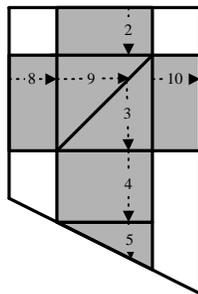


Figure 11 Implied relaxed swath segment selection problem solution

Given the network flow relaxation, we are tempted to simply use this as our heuristic reward estimator for a branch and bound search, but this would be a mistake. As soon as a segment is selected during search, the flow network needs to be adjusted to reflect the lost capacity due to the segment selection. Often, waste is associated with this allocation. We have implemented an incremental flow network capacity update that allows us to change the capacities upon segment selection and de-selection. The time complexity of the update is proportional to the total

number of depot's and the total number of elements associated with the segment being updated.

Now we have a good heuristic reward estimator that we can apply to a traditional branch and bound search. We need a node ordering heuristic that takes a partial solution and the search options available and orders the options accordingly, hoping to find good solutions early. Specifically, we need to take the partial solution \mathbf{R}' and consider which segments $s \in \mathbf{R}, s \notin \mathbf{R}'$ to include. The basic approach is to calculate the reward/cost for including each set not yet included. It is the score for selecting a segment is the sum of the reward of each of the shards covered by the segment that have not yet been collected divided by the capacity cost of the segment. Ties are broken randomly.

Search ensues thusly:

1. Let $b \leftarrow 0$, i.e., the current best quality bound
2. Let $\mathbf{R}' \leftarrow \emptyset$, i.e., our current best solution
3. Let $\mathbf{P} \leftarrow \emptyset$, i.e., our current partial solution
4. Let $reward(\mathbf{P})$ be the summed reward of the shards collected in the partial solution \mathbf{P}
5. Let $OpenList$ be a priority queue of segments where priority is based on reward/cost ratio gain given the partial solution \mathbf{P} . $pop(OpenList)$ returns the highest valued segment while removing it from $OpenList$
6. Let $OpenList \leftarrow S$
7. Let h be a real-valued heuristic function that returns the quality of the network flow relaxation of the remaining segments in $OpenList$ given the partial solution \mathbf{P}
8. $search(\mathbf{R}', \mathbf{P}, b)$

Recursive search routine

- ```

search($\mathbf{R}', \mathbf{P}, b$)
9. if $reward(\mathbf{P}) + h < b$ return, i.e., prune
10. if $OpenList = \emptyset$, i.e., the bottom of the recursion
11. if $reward(\mathbf{P}) > b$
12. $\mathbf{R}' \leftarrow \mathbf{P}$
13. $b \leftarrow reward(\mathbf{P})$
14. end if
15. return
16. end if
17. $s \leftarrow pop(OpenList)$
18. if feasible to insert s into \mathbf{P}
19. insert s into \mathbf{P}
20. search($\mathbf{R}', \mathbf{P}, b$)
21. remove s from \mathbf{P}
22. end if
23. search($\mathbf{R}', \mathbf{P}, b$)
24. push s back onto $OpenList$

```

### 3 Results

We report "first solution" time and quality results for ASTER, BnBFlow, and Integer Programming for many sizes of random problems. Problems are randomly generated SSSPs.

Easily computable metrics that appear to reflect on the scale of the problems and the quality of solutions are the number of shards in  $H$  for each problem and the initial network flow approximation, thus we report these for the sizes of problems here. We report results for 100 instances per size, with a time cutoff of 1 hour. It is important to note that ASTER required less than 1 second for any instance.

Figure 12 compares a solution for a typical SSSP solved using ASTER and BnBFlow. Shading indicates the solution area.

Figure 13 shows a comparison of BnBFlow (our technique using branch and bound with a network flow heuristic) and IP (Integer Programming; the “straw-man” for optimal solutions). Times are for optimal solutions, except where time is two hours. Clearly, BnBFlow took less time in finding the optimal solutions where both found optimal solutions.

Figure 14 compares BnBFlow, ASTER, and IP for solution quality. Note that we also include the network flow value as an upper bound on quality—Relaxation. This

is not so important for those values where optimal values are found, but is good for comparing values where neither algorithm found optimal value, e.g., shard counts greater than 8000.

ASTER dominates in terms of generating a fast solution. But the time cost of BnBFlow is minimal compared to the solution quality. Integer programming returns an optimal solution, but does not outperform BnBFlow, and for relatively small problems doesn’t terminate. Thus, in terms of any-time performance, the best strategy appears to be to first use ASTER followed immediately by BnBFlow. (See Figure 13 and Figure 14.) Note that the quality of the BnBFlow solutions continues to track the quality of the Relaxation, indicating good any-time performance by the technique.

But, as mentioned earlier, in the case that the reward for a member of a set is not proportional to the capacity cost of a set, then IP is the stronger formulation.

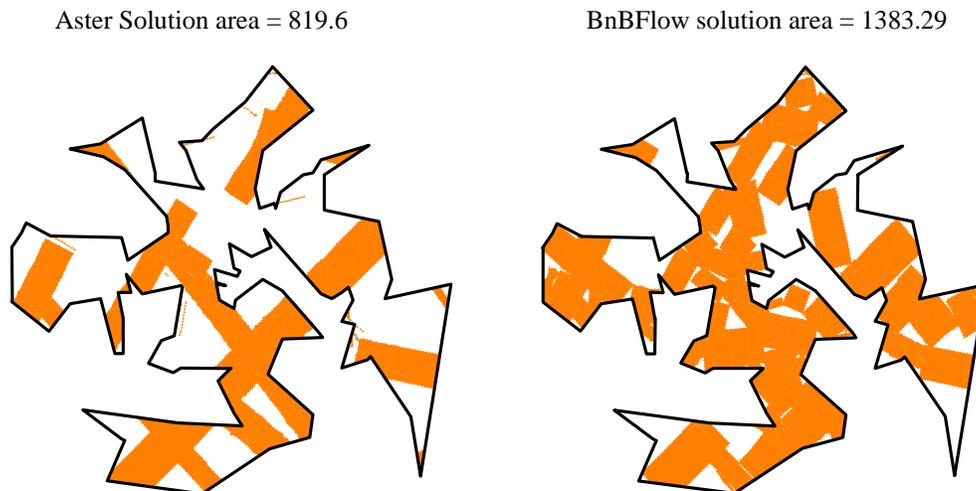


Figure 12 ASTER and BnBFlow comparison

## Solution Time by Problem Size

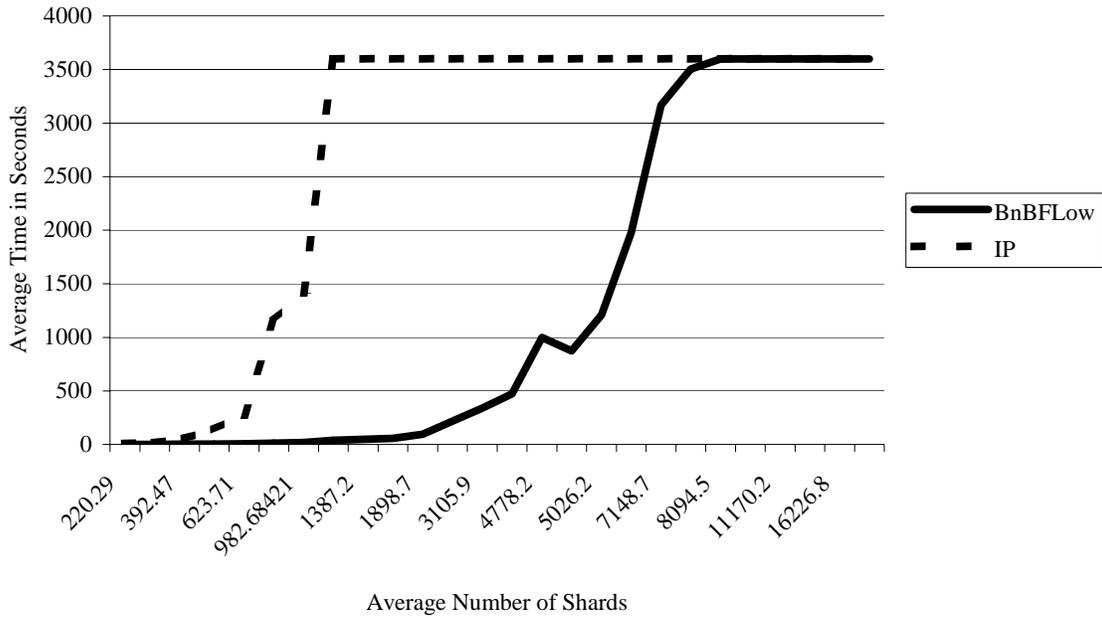


Figure 13 Comparative time performance

## Solution Quality by Problem Size

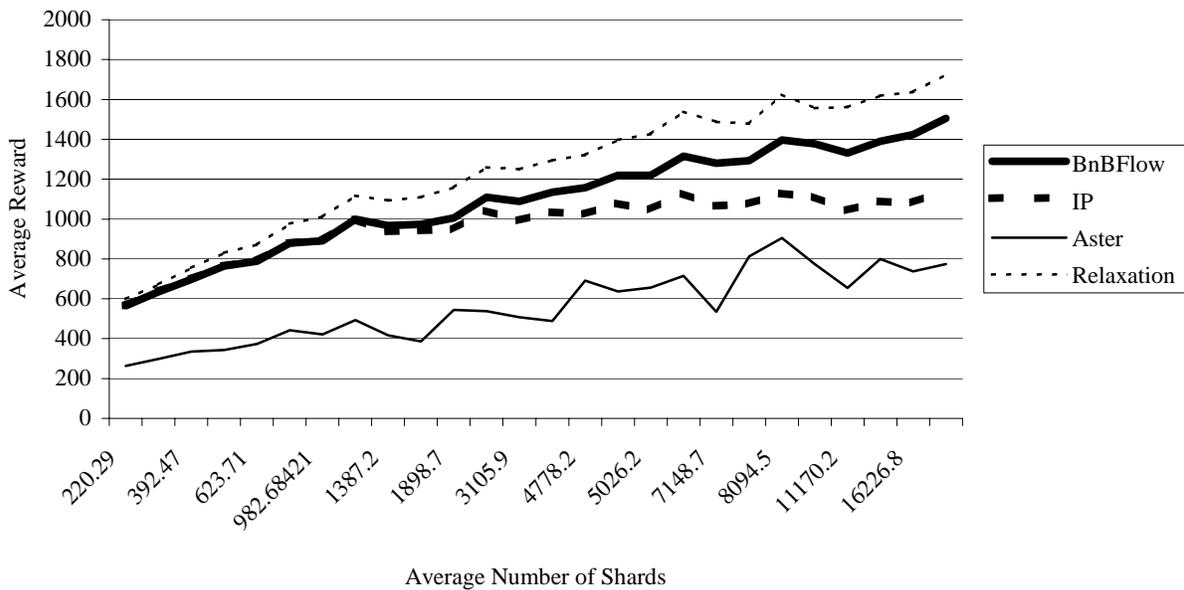


Figure 14 Comparative quality performance

## 4 Related Work

The best previous problem solver for SSSPs is the ASTER system [Muraoka 1998]. They use greedy maximization, breaking ties by choosing earliest segments first, to find a solution. Their algorithm is deterministic and very fast. Basically, they subdivide the problem into separate legs, and schedule each leg. As it turns out, a leg corresponds to a day of operations. For each leg, they include the segment that has the best reward/cost value, until no more segments can be accommodated. They break ties by choosing the earlier segment.

Work on a somewhat similar problem with more degrees of freedom is reported by [Frank 2000]. In this case, a route for an aircraft-borne off zenith observatory must be planned that maintains pointing at celestial targets over an interval of time. The route is flexible (as opposed to our fixed routes) and more constraints (maximum fuel usage, round trip travel, etc.) are considered, but on-board memory is not a prohibitive factor.

For a good example of a polyhedral solution to a combinatorial optimization problem having to do with satellite scheduling (formulated as a pick-up and delivery problem), see [Ruland 1986].

[Oddi 2003] solves a constrained-memory domain with fewer types of constraints called the Mars Express Memory Dumping Problem. The system uses a portfolio approach to solving the problem as formulated in a constraint-based framework. The portfolio consists of a tabu search strategy, a random sampling strategy, and a greedy strategy.

More general constraint-based frameworks for scheduling that have been applied to spacecraft operations include that of [Dungan 2002], [Ghallab 1994], and [Chien 2000]. In each of these, the problem is expressed as a set of constraints to be satisfied. In the case of [Dungan 2002], and [Ghallab 1994], the systems search the feasible space of domains in the constraint space. In the case of [Chien 2000] the system searches both the infeasible and feasible space of value assignments, using randomized local search.

## 5 Acknowledgements

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not

constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

## Appendix A SSSP NP-completeness proof

The SSSP is NP-complete. It is contained by NP in that we can “guess” a solution (the set of swaths segments to take) and verify our guess in deterministic polynomial time. It contains NP in that any instance of a number partitioning problem can be reduced to an equivalent instance of the SSSP. (Number partitioning is NP-complete.) The reduction follows.

As a reminder, number partitioning is, given a set of positive integers  $V$ , partition  $V$  into two subsets  $V1$  and  $V2$  that do not intersect, yet contain all elements of  $V$  in their union. The sum of the elements of  $V1$  and  $V2$  must be equal.

The idea underlying our transformation is to determine  $V1$  by forcing a selection that just exactly fits into  $V1$ . Our transformation is as follows.

$$\sum_{i=1}^{|V|} V_i$$

Let  $c = \frac{\sum_{i=1}^{|V|} V_i}{2}$ , i.e.  $c$  is the target sum for  $V1$  (and  $V2$ ).

Let  $D = \{d\}$ , with  $cap(d) = c$ .

$\forall v \in V$ , add a shard  $h$  to  $H$ , i.e.,  $reward(h) = V_i$ .

$\forall h \in H$ , add a segment  $s$  to  $S$ ,  $cap(s) = reward(h)$ ,  $shards(s) = \{h\}$ .

Let  $\mathbf{R} = S_1, S_2, \dots, S_{|S|}, d$

Having the downlink, shards and segments, solve the SSSP. If the SSSP has a value of  $c$ , then a perfect partition exists. For example, if  $V = \{1, 2, 4, 10, 15\}$ , then

$$\begin{aligned} D &= \{d\}, \\ cap(d) &= 16, \\ H &= \{\alpha, \beta, \chi, \delta, \epsilon\}, \\ reward(\alpha) &= 1, \\ reward(\beta) &= 2, \\ reward(\chi) &= 4, \\ reward(\delta) &= 10, \\ reward(\epsilon) &= 15 \end{aligned}$$

$S$  has a one-to-one relationship with  $H$ , with

$$\begin{aligned} cap(S_1) &= 1, shards(S_1) = \alpha, \\ cap(S_2) &= 2, shards(S_2) = \beta, \\ cap(S_3) &= 4, shards(S_3) = \chi, \\ cap(S_4) &= 10, shards(S_4) = \delta, \\ cap(S_5) &= 15, shards(S_5) = \epsilon \end{aligned}$$

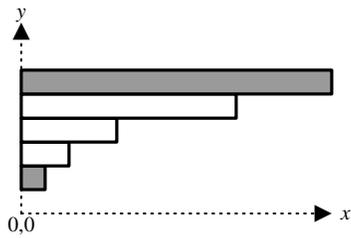


Figure 15 Number partitioning to SSSP example  
One solution to this instance is  $R' = S_1, S_5, d$ , as illustrated in Figure 15 (shaded area is the solution area).

## References

- [Chien 2000] S. Chien, G. Rabideu, R. Knight, R. Sherwood, B Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, and D. Tran. "Automating space mission operations using automated planning and scheduling." In *Proc. SpaceOps*, 2000.
- [Corman *et al*] Thomas H. Corman, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.
- [Dungan 2002] J. Dungan, J. Frank, A. Jonsson, R. Morris, and D. Smith. "Advances in Planning and Scheduling of Remote Sensing Instruments for Fleets of Earth Orbiting Satellites." *Earth Science Technology Conference*, 2002. Pasadena, California.
- [Frank 2000] J. Frank. "SOFIA's Choice: Automating the Scheduling of Airborne Observations" Proceedings of the 2d NASA Workshop on Planning and Scheduling for Space, March 2000.
- [Garey and Johnson 1979] M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, San Francisco, 1979.
- [Ghallab 1994] M. Ghallab and H. Laruelle, "Representation and control in IxTeT, a temporal planner", in *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, pp. 61-67, Chicago, IL, (1994). AAAI Press, Menlo Park.
- [Karp 1972] R. M. Karp "Reducibility among combinatorial problems." In R. E. Miller and J. W. Thatcher (eds.) *Complexity of Computer Computations*, Plenum Press, New York, 85-103.
- [Muraoka 1998] H. Muraoka, R. H. Cohen, T. Ohno, and N.Doi. "ASTER Observation Scheduling Algorithm." *SpaceOps 98*. 1998, 1-5 June, Tokyo, Japan.
- [Oddi 2003] A. Oddi, N. Policella, A. Cesta and G. Cortellessa. "Generating High Quality Schedules for Spacecraft Memory Downlink Problems." *Ninth International Conference on Principles and Practice of Constraint Programming*, 29 September - 3 October, 2003, Kinsale, County Cork, Ireland.
- [Papadimitriou and Steiglitz 1982] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982.
- [Ruland 1986] K. Ruland. Polyhedral solution to the pickup and delivery problem. Washington University, Sever Institute of Systems Science and Mathematics. <http://rodin.wustl.edu/~kevin/dissert/dissert.html> (Dissertation). St. Louis Missouri, 1995.
- [Schrijver 1986] A. Schrijver. *Theory of Linear and Integer Programming*, Wiley, 1986.
- [Zhang 2000] W. Zhang. "Depth-First Branch-and-Bound versus Local Search: A Case Study." In *Proceedings of the 17<sup>th</sup> National Conference on Artificial Intelligence (AAAI 2000)*, pages. 930-935, Austin, Texas, 2000.