

# Solving Large Scheduling Problems by Coordinating Specialized Sub-Problem Solvers

**Russell Knight and Ben Smith**

Jet Propulsion Laboratory  
California Institute of Technology  
Pasadena, CA 91109-8099  
{firstname.lastname}@jpl.nasa.gov

## Abstract

NASA has a need for planning and scheduling systems that are expressive enough to encode real-world problems, flexible enough to keep up with shifting problem requirements, and powerful enough to provide high quality solutions within reasonable time bounds. These competing demands are particularly difficult to meet for the many NASA planning and scheduling problems that contain interacting combinatorial optimization sub-problems.

General purpose planners are expressive and flexible, but perform poorly on these complex problems. Combinatorial optimization solvers have excellent performance, but are only applicable to sub-problems. This paper describes a simple method for integrating certain kinds of combinatorial sub-problem solvers within a general purpose planner/scheduler framework that demonstrably increases solution speed and quality.

## Introduction

Many NASA observation scheduling problems are quite large, often containing hundreds or even thousands of observations. It is difficult to find high quality solutions to these problems within reasonable time bounds. The problems are often too large and complex to make globally optimal solutions practical, and even good locally optimal solutions can be computationally expensive. Problem-specific scheduling algorithms can exploit the problem structure to find high quality solutions more quickly, but they are expensive to develop and must be redesigned if the problem specification changes. This is a real concern: the problem specification often

evolves throughout the mission life cycle as the problem and mission trade space become better understood.

General purpose scheduling algorithms can accommodate evolving problem specifications, but perform poorly on these hard problems because they have little knowledge of the problem structure.

This paper discusses a middle-ground approach that improves the performance of flexible general purpose schedulers by identifying the combinatorial optimization sub-problems and coordinating specialized solvers for these sub-problems.

For example, combinatorial sub-problems that occur in NASA observation scheduling problems include:

- schedule a set of observations given constraints when they are visible and minimum slew intervals between adjacent observations (*traveling salesperson problem with time windows*)
- Assigning observations to fixed downlink opportunities (*bin packing or k-knapsack*)
- Select minimum number of observations that will cover of a region-of-interest (*set covering*)

Several of these sub-problems may occur within a single observation scheduling problem, along with other resource and operations constraints. This paper refers to a scheduling problem that contains interacting combinatorial sub-problems as a “composite”

problem, and refers to the individual sub-problems as “elemental” problems.

One straightforward method for incorporating these sub-problem solvers is to simply join the search control rules for each of the sub-problems. This approach can be employed when all of the problems can be expressed in a single search formalism such as iterative repair (IR) or integer programming (IP).

In the remainder of this paper we define a scheduling problem with interacting sub-problems (TSP and bin-packing), encode it in a general purpose scheduler, and discuss how to incorporate specialized solvers for the two sub-problems. The performance of this method is then compared to two alternate solution methods: integer programming and iterative repair with domain-independent heuristics.

The IR approach with integrated sub-problem solvers yields superior anytime performance over the ‘practical’ region of the performance vs. time curve, and the IP solver dominates the optimal region of the curve. The IR approach has the added advantage that it integrates more easily with general purpose planning & scheduling systems than does IP.

### **An Observation Scheduling Problem**

A common NASA scheduling problem is observation scheduling. The objective is to schedule as many observations as possible from an oversubscribed list, subject to constraints on target visibility and onboard data storage capacity. The acquired data must be downlinked during time intervals when a ground receiving station is visible and available. There may also be setup times and mode constraints that depend on adjacent observations.

For purposes of this paper we identify a specific instance of the observation scheduling problem derived from the Space Interferometry Mission. This example will be

used throughout the paper to ground the discussion and evaluate the methods described here. We refer to this problem as the *tiling problem*, for reasons that will become clear below.

One of SIM’s primary objectives is to acquire interferometric measurements of the celestial sphere. Each observation covers a very small region of the sky, called a “tile”. To observe the entire celestial sphere, each of over a thousand overlapping tiles must be observed for about half an hour in two different orientations. Slewing the spacecraft between adjacent tiles takes several minutes, and proportionally longer between distant tiles. The observation campaign must be completed within as short a time as possible to maximize science return.

This much of the problem is a traveling salesperson problem (TSP) in which the tiles are cities and the cities are uniformly distributed over a sphere. However, there are additional constraints that complicate the problem. To avoid instrument damage the instrument must not point within several degrees of bright bodies such as the Sun, Jupiter, Saturn, and the Earth. These move over the course of the campaign, which imposes several windows during which tile is not observable. This makes it a TSP with multiple time windows.

The spacecraft has limited onboard memory, so the data must be downlinked, but only during certain times when a ground receiving station is visible. Allocating observations to downlink windows can be expressed as a bin packing problem.

### **Formal Specification of the Tiling Problem**

An instance of a tiling problem describes the observations to be made and the opportunities for downlinking the observation data. A solution to a specific tiling problem consists of a start-time for

each observation or an indication that the observation is to be ignored.

What follows is a formal description of the tiling problem, as well as a proof that the tiling problem is NP-complete. The proof is based on a reduction of the Hamiltonian cycle problem to the tiling problem.

More formally, an instance of a tiling problem  $TP$  is an 6-tuple  $\langle V, D, t, w, c, s \rangle$  where  $V$  is a set representing observations,  $D$  is a set representing downlinks,  $t$  is a function that maps each  $v \in V$  to a set of non-overlapping intervals (the time-windows of an observation),  $w$  is a function that maps each  $v \in V$ , each  $d \in D$ , and each pair of observations  $(v_1, v_2)$ ,  $v_1$  and  $v_2 \in V$  to a positive real value (duration: either duration of a downlink, duration of an observation, or duration to traverse from one observation to another),  $c$  is a function that maps each  $v \in V$ , and each  $d \in D$  to an integer (capacity: either the amount memory required to store an observation or the capacity of a downlink), and  $s$  is a function that maps each  $d \in D$  to the positive real numbers (start time of a downlink).

It is assumed that downlinks and their associated durations do not overlap temporally. Interesting aspects of the tiling problem that set it apart from other problems are 1) every downlink is used, 2) data cannot be carried over a downlink (all data is downlinked or erased), and 3) observations can be skipped. Thus, we can reformulate an TP  $T$  as an equivalent TP  $T'$  where we need not consider any  $w(d \in D)$  or  $w(v \in V)$  by adjusting the associated time-windows  $t(v \in V)$  and adjusting the durations  $w(v_1, v_2 \in V)$ . Therefore, we consider  $t(v \in V)$  to be a set of intervals that represent possible start-time assignments for  $v$ .

A solution is a 2-tuple  $\langle I, st \rangle$  where  $I$  is a subset of  $V$  representing ignored observations (ignore-list), and  $st$  is a function that maps

each  $v \in V-I$  to the real numbers representing the assignments of start-times for the observations such that they can be accommodated by the downlinks. A capacity-optimal solution is a 2-tuple  $\langle I, st \rangle$  that maximizes the sum of data actually downlinked. A cardinality-optimal solution is a 2-tuple  $\langle I, st \rangle$  that maximizes the number of observations in  $V-I$ . The associated decision problem would be a TP with a bound  $b$ , where  $b$  is the minimum amount of data downlinked for success of a summed-capacity-decision problem (TPSD), or  $b$  is the minimum number of observations required for success of a cardinality-decision problem (TPCD).

### The Tiling Problem is NP complete

**Proof 1:** *Both the TPSD and the TPCD are NP-complete.* Each problem is clearly contained by NP in that a solution can be guessed in polynomial time (i.e., guess  $|I|$  observations to ignore and  $|V-I|$  start-times for a total of  $|V|$  guesses) and verified (i.e., ensure all  $|V-I|$  start-times fall within a valid window; sum the appropriate  $c(v)$  values for each downlink  $d \in D$  and compare to the capacity of the downlink  $c(d)$ , and either sum  $c(d)$  values or take the cardinality of  $I$  to evaluate the decision).

**Proof 2:** *Both the TPSD and the TPCD are NP-hard.* Every instance of a Hamiltonian cycle problem (HCP), which is NP-hard, can be reformulated as a TPCD or TPSD as follows. An instance of a HCP is a graph  $(V, E)$ . A solution is a simple cycle that contains that contains all  $v \in V$ . Thus, we must map a HCP  $\langle V, E \rangle$  to a TP  $\langle V, D, t, w, c, s \rangle$  and show that the use of decision criteria for TPSD and TPCD do not affect the transformation. To map HCP( $V$ ) to TP( $V$ ), we choose a vertex in HCP( $V$ ) and call it  $\alpha$ . We also create use a duplicate vertex to  $\alpha$  called  $\omega$ . Thus  $TP(V) = \{\alpha, \omega\} + HCP(V) - \{\alpha\}$ . Henceforth, TP( $V$ ) is referred to as  $V, D$

contains a single downlink  $d$ . For each  $v \in V$ , if  $v = \alpha$ ,  $t(v) = \{(1,1)\}$ , else if  $v = \omega$ ,  $t(v) = \{(|V|,|V|)\}$ , otherwise  $t(v) = \{(2, |V|-1)\}$  (i.e., any time assignment before the downlink is ok, with alpha coming first and omega coming last). For each  $v_1 \in V$ ,  $v_2 \in V$ ,  $w((v_1, v_2)) = 1$  if  $(v_1, v_2) \in E$ ,  $\infty$  otherwise. Note that  $\alpha = \omega$  with respect to  $E$ . For the only  $d \in D$ ,  $c(d) = |V|$ ; for each  $v \in V$ ,  $c(v) = 1$  (i.e., the only downlink has the capacity to handle all observations). Note that the sum of all  $v \in V-I$  for a solution  $I$  is the same as the cardinality of  $V-I$ , thus the objective function for both the TPSD and the TPCD are equivalent for this formulation. By Lemma 1, a solution exists for the HCP only if a solution of value  $|V|$  exists for the TPSD (or TPCD) making these problems NP-hard. Therefore, the TPSD and the TPCD are NP-complete.  $\square$

**Lemma 1:** A solution exists for the HCP only if a solution of value  $|V|$  exists for the TPSC (or TPCD). We can convert any solution  $\langle I, st \rangle$  to a TP into a solution  $\langle v_1, v_2, \dots, v_n \rangle$  of an HCP. We assign  $\alpha = \omega = v_1$  of the permutation. Now, for each  $v \in TP(V)$ , we assign the permutation order  $vst(v)$ . There must exist a path from  $\alpha$  through all other vertices and terminating at  $\omega$ , otherwise there must exist non-adjacent vertices  $v_a$  and  $v_b$  that are adjacent in the permutation, but this would require a difference between the start-times of  $1 = st(v_b) - st(v_a) = w((v_a, v_b)) = \infty$ , a contradiction.

### Integer Programming Formulation

We give a formulation of the TP as a mixed integer/linear program (MIP) using the standard notation of variables being a vector  $x$ , and constraints being linear inequalities on  $x$ . This formulation is based on the TSP formulation of Grötschel & Holland (1988) and the bin-backing formulation of Padberg (1979).

#### Variables:

For each observation  $v \in V$ ,  $x(stv)$  is the assigned start-time for observation  $v$ .  $x(stv)$  is a continuous, real-valued variable. These are the start-time variables.

For each observation  $v \in V$ , for each allowed time interval (window)  $(z, y)$  of  $t(v)$ ,  $x(t_vzy)$  is an integer binary value that is 0 if  $x(stv)$  should be contained in the interval  $(z, y)$ , and 1 if not. These are the time-window assignment variables.

For each pair of observations  $(v_1, v_2)$  where  $v_1$  and  $v_2 \in V$ ,  $x(wv_1v_2)$  is an integer binary value that is 0 if  $x(stv_1) >$  all other  $x(stv) \mid x(stv) < x(stv_2)$  and 1 otherwise (i.e.,  $v_1$  immediately precedes  $v_2$ ). These are the observation-adjacency variables.

#### Constraints:

For each  $v \in V$ , the sum of all  $x(t_vzy) = |V| - 1$  (i.e., only one observation assignment to all of its associated time-windows is allowed). These are the time-window unit constraints.

We assume a value  $b$  that is greater than any possible assignment to start-times including the case if all start-times occur after the last downlink. For each  $x(wv_1v_2)$ ,  $x(wv_1v_2)b - x(stv_1) + x(stv_2) \geq w(v_1, v_2)$ . These are the observation adjacency constraints.

For each  $x(t_vzy)$ ,  $x(t_vzy)b + x(stv) \geq z$ , and  $-x(t_vzy)b + x(stv) \leq y$  (i.e., the start-time of an observation must be contained by its chosen time-window and be ignored by all other time-windows). These are the start-time containment constraints. Note that it is perfectly feasible to ignore all time-windows; it is the role of the objective function to enforce that as many observations are assigned (within its criterion) as possible.

We assume a mapping of time-windows of an observation  $v$  to each downlink  $d \in D$ . This set of time windows is referred to as  $t(v, d)$ . For each downlink  $d$ , add the constraint

$$\sum_{v \in V} \sum_{(x,z) \in t(v,d)} x(t_{v,z})c(v) \geq -c(d) + \sum_{v \in V} |t(v,d)c(v)|,$$

(i.e., the skipped capacity must be no less than the total possible skipped capacity minus the downlink capacity for any given downlink.) These are the downlink capacity constraints.

### Iterative Repair Formulation

For this solution approach the tiling problem was encoded in the ASPEN planning and scheduling framework (Chien *et al.*, 2000). The elements in the ASPEN domain modeling language are activities, states, resources, and constraints. An activity is an action the spacecraft can perform, such as an observation or downlink. Activities have a start time and duration and may overlap each other. A resource represents a physical or logical resource of the spacecraft, such as the onboard memory. A state represents a physical or logical state of the spacecraft, such as the spacecraft attitude or whether a given ground station is visible. Each state and resource is represented as a *timeline* that shows how it evolves over time.

The activities, states, and resources are related by *constraints*. Each activity instance imposes constraints that must be met whenever that instance is in the plan. These can be temporal constraints among activities, resource constraints (e.g., an observation uses  $d$  seconds of onboard storage tape, where  $d$  is the duration of the observation), and state constraints (the ground station must be visible during a downlink).

The tiling problem was encoded as follows. The activities are *observe(target)*, *downlink*, and *slew(a,b)*; the state is *attitude*; and the resource is *onboard memory*. An *observe(target)* activity consumes onboard memory and requires that the *attitude* state be equal to *target*. The *downlink* activity restores onboard memory and requires that the attitude be `GROUND_STATION`. The *slew(A,B)* activity

requires that the *attitude* is *A* just before the activity and *B* after the activity. The duration of the *slew* activity is the time it takes to slew the spacecraft between attitudes *A* and *B*.

An instance  $\langle V, D, t, w, c, s \rangle$  of the tiling problem is expressed in ASPEN by one *observe* activity instance per observation in  $V$ , one *downlink* activity per opportunity in  $D$ . The duration and start time of the downlink windows are specified by  $w$  and  $s$  respectively, and the duration and start time of the observation windows are similarly specified by  $w$  and  $v$ . The duration of *slew(A,B)* is specified by  $w: \forall xV \rightarrow \text{positive integers}$ . A solution consists of a subset of *observe* activities that satisfies all of the constraints. The objective function is a weighted sum of the activity score (higher is better) and the makespan (smaller is better).

### Iterative Optimization Search Algorithm

The tiling problem is solved by an iterative optimization algorithm (Rabideau *et al.*, 1999). At each step in the search we have a schedule that might violate some or all of the constraints. Iterative repair analyzes each of these violations, selects one, and performs operations on the schedule to remove the violation. Repair operations include reassigning an observation's start time and moving it to the ignore-list. A given repair may lead to more violations, which are handled similarly.

Once a valid schedule is found, iterative optimization performs operations on the schedule that improve one of the *preferences*. A preference is an element of the objective function, such as "minimize the number of ignored observations", "maximizing the amount of downlinked data", and "minimize makespan". The objective function is a weighted sum of these preferences. Optimization operations include removing observations from the ignore-list and

selecting an earlier start time for an observation.

### **Integrated Solvers Formulation**

The iterative optimization method can be improved by exploiting some knowledge of the problem structure. The tiling problem contains two interacting combinatorial optimization sub-problems: TSP with time windows (for finding a minimum-makespan tour through the celestial sphere) and bin packing (for assigning observations to downlink opportunities).

One way to provide this knowledge is to analyze the problem and develop a specialized search algorithm or problem-specific heuristics. However, these can be expensive to develop and are brittle to changes in the problem formulation. An alternate approach that we explore here is to identify the combinatorial optimization sub-problems and employ control knowledge from existing solution algorithms that have been developed over decades of study in the research community.

A key question is how to coordinate the two solvers. Because the TSP and bin-packing problems interact, a high quality solution to one problem may be incompatible with high quality solutions to the other; and both sub-problems may interact with additional constraints in the overall problem.

We identified local-search versions of the bin packing and TSP algorithms, and added their control rules into the iterative repair scheduler for the overall problem. The iterative repair framework then decides which rules to apply in any given step by choosing the rule that leads to the largest local gain in feasibility or quality. The sub-problem heuristics guide the overall problem toward high quality solutions to the sub-problems, and conflicts between the sub-problems solutions are arbitrated by

evaluating their impact on the overall problem.

This is an admittedly unsophisticated method, yet it performs quite well. It is also flexible to changes, in that the heuristics can be easily modified as new sub-problems are added, removed, or changed. Future work will investigate more sophisticated algorithms and compare them to this baseline performance.

The TSP heuristic makes use of insertion algorithms (choosing randomly among either random insertion, insert furthest, or greedy insertion). These techniques give good TSP performance with light computation. The TSP heuristics also make use of 2-opt swaps (Hochbaum 1997)

The bin packing solver utilizes the “biggest-first” algorithm, which first orders the items from largest to smallest, then places them sequentially in the first bin in which they fit. This strategy is always within approximately 22% of optimal, and no strategy can guarantee performance better than 22% of optimal unless  $P=NP$  (Hoffman 1998).

### **Performance Results**

Empirical performance results are shown in Table 1. Each of the three solution methods (uninformed iterative repair, uninformed integer programming, and informed iterative repair) solved the same 100 random instances of the tiling problem. Each problem instance had a pool of 50 observations to be scheduled, an average of 25 windows of availability per observation, and 16 downlinks. The actual number, start time, and duration of the visibility windows and downlink windows for the 100 instances were randomly generated according to a normal distribution about these mean values.

The reported value for the schedule quality (score) is normalized against the maximum possible score if all observations were

scheduled. This ensures, for example, that perfect schedules for problem instances with different global maxima would be reported as being of equal quality.

The running times reported are for a Sun Ultra 2 computer. The base for the temporal log scale is 1.5.

### **Discussion**

Integer programming produces the highest quality results, but only for running times over 15 hours. For running times between 10 seconds and 15 hours the dominant method is iterative repair with incorporated bin-pack and TSP solvers. For running times under 10 seconds iterative repair produces the best solutions, though informed iterative repair is a close second.

Uninformed iterative repair does not require knowledge of the problem structure, and utilizes only problem-independent heuristics. It performs what is essentially a randomized local search, where the neighborhood operators improve either feasibility or quality. This means that it does not need to be “tuned” if the problem changes, but that same lack of knowledge negatively impacts performance.

For the tiling problem, uninformed iterative optimization quickly finds a feasible solution and improves it to a local optima. However even with very long running times the local repair moves cannot improve much on that local optima. Performance quickly asymptotes to a middling quality solution.

Iterative repair informed by sub-problem heuristics performs much better. The sub-problem heuristics guide the overall problem out of local minima and towards high quality solutions to the sub-problems. However, the best overall solution may require sub-optimal solution to the sub-problems. Conflicts between the sub-problems solutions are arbitrated by evaluating their impact on the overall problem. Although this is admittedly unsophisticated, it performs well. In fact, it

dominates the optimal IP solver for running times under 15 hours.

The performance results are for problem size of fifty observations. The full tiling problem can have over a thousand observations. To obtain high quality solutions to these large problems within reasonable time bounds, methods that dominate at smaller run times are clearly preferable. Of the three, iterative repair with sub-problem heuristics performs best over this ‘practical’ region of the time vs. performance curve.

### **Conclusions**

Many real-world NASA scheduling problems require good “practical” performance—that is, high quality but not necessarily optimal solutions that can be obtained within reasonable computational resources. The solution algorithms must also be flexible to changes in the problem formulation. That is, incremental changes to the problem should require small inexpensive changes to the encoding and result in similar performance without costly redesign of the solution algorithm.

Monolithic IP solvers are one common approach for solving these kinds of scheduling problems. Although they provide high quality or even optimal solutions, they have several limitations: they are inflexible to changes in the problem specification, and good performance requires heuristics derived from deep analysis of the problem structure—often a significant undertaking.

Iterative repair appears to provide better practical performance and greater flexibility. Uninformed iterative repair outperforms uninformed integer programming over the practical region of the performance curve. However, neither uninformed approach is fast enough for large problems such as the tiling problem.

Local search sub-problem solvers can be integrated within an iterative-repair planner/scheduler by simply taking the union of the local search rules. Exploiting this sub-problem knowledge improves performance considerably for the iterative repair solver, enabling it to break out of local minima. This approach also preserves flexibility: when the problem changes, it is relatively easy to identify sub-problems and introduce control knowledge as compared to doing a full-up analysis of the problem structure.

Overall this work indicates that high quality solutions to large scheduling problems can be obtained within reasonable computational resources if (a) the problem contains combinatorial optimization sub-problems and (b) good local-search solution algorithms exist that make few if any assumptions about the global problem structure.

### Future Work

The solver integration methods discussed in this paper are limited: the solvers must rely on local assumptions about the problem structure, and it is susceptible to getting trapped in local minima because negative interactions among solvers are resolved locally (biggest gain to the objective function wins). More sophisticated methods are needed to overcome these limitations. Our future work will investigate such methods in order to improve on the performance results reported here.

### Acknowledgements

This paper describes work performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract from the National Aeronautics and Space Administration.

### References

- Chien, S.; Rabideau, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F.; Barrett, T.; Stebbins, G.; and Tran, D. (2000). ASPEN—Automating Space Mission Operations using Automated Planning and Scheduling. In *SpaceOps 2000*. Toulouse, France.
- Crowder, H. & Padberg, M. (1980). Solving large-scale symmetric traveling salesman problems to optimality, *Management Sci.* **26**: 495-509.
- Du, D.-Z. and Pardalos, P.M. (Eds.) *Handbook of Combinatorial Optimization*. (1998) Kluwer Academic Publishers.
- Grötschel, M. & Holland, O. (1988). Solution of large-scale symmetric travelling salesman problems, Technical Report 73, Institut für Mathematik, Universität Augsburg.
- Hochbaum, D. *Approximation Algorithms for NP-Hard Problems*. Boston: PWS Publishing, 1997
- Hoffman, P. *The Man Who Loved Only Numbers: The Story of Paul Erdos and the Search for Mathematical Truth*. p. 172, New York: Hyperion, 1998.
- Padberg, M. (1979). "Covering, packing and knapsack problems," *Mathematical Programming* **47**, 19-46.
- Miller, C. E., Tucker, A. W. & Zemlin, R. A. (1960). Integer programming formulations and the traveling salesman problem, *J. Assoc. Comput. Mach.* **7**: 326-329.
- Nemhauser, G. L. & Wolsey, L. A. (1988). *Integer and Combinatorial Optimization*, John Wiley, Chichester, UK