To appear in 10th International Workshop on Planning and Scheduling for Space (IWPSS 2017) 15-16 June 2017

A Hybrid Traveling Salesman Problem - Squeaky Wheel Optimization Planner for Earth Observational Scheduling

Garrett Lewellen and **Christopher Davies** and **Amos Byon** and
Russell Knight and **Elly Shao** and **Daniel Tran** and **Michael Trowbridge**
Jet Propulsion Laboratory, California Institute of Technology
{christopher.j.davies, amos.j.byon, russell.l.knight, elly.j.shao,
daniel.q.tran, michael.a.trowbridge}@jpl.nasa.gov

Abstract

We outline a hybrid planner for scheduling Observation Requests on an Earth observing satellite, subject to a variety of constraints for the ASPEN (Chien et al. 2000) Eagle Eye adaptation (Knight, Donnellan, and Green 2013) that combines Squeaky Wheel Optimization (Joslin and Clements 1999) with sliding observation planning (Aldinger et al. 2013). The Earth Observing Satellite (EOS) planning problem (Globus et al. 2004) is reformulated as time-varying travel time TSP with interval constraints (Ichoua, Gendreau, and Potvin 2003). The replanning/fill stage of the hybrid scheduler marginally improves schedule quality for all bus agilities examined, but has more impact on lower agility observers. The squeaky wheel stage primarily affects overall schedule quality by satisfying high priority requests, while the replanner reduces starvation of lower value requests.

Introduction

Eagle Eye is an observational spacecraft planning and scheduling system built atop ASPEN (Chien et al. 2000). A user specifies targets to be observed, and Eagle Eye generates a schedule of instrument slews and pointings to fulfill these requests. The optimal observation schedule is a largest value tour within a graph that has bidirectional edges with asymmetric (figure 12), time-varying slew cost edge weights (figure 1), cycles (revisits) and interval constraints. Similar problems have been classified as NP-hard and NP-complete (Karger, Motwani, and Ramkumar 1997; Lemaître et al. 2002; Ichoua, Gendreau, and Potvin 2003; Pinedo 2012).

Science teams will always desire an optimal schedule, but combinatorial runtime is unacceptable for a real scheduler. We present a compromise in this paper – a hybrid approximation algorithm that uses squeaky wheel optimization for initial scheduling and a looped sliding window/fill replanning scheme to improve schedule quality. We offer a weaker guarantee that each iteration will be as good as or better than the prior iteration, but do not guarantee optimality.

Copyright © 2017, by the California Institute of Technology. ALL RIGHTS RESERVED. United States Government Sponsorship acknowledged.

Contact author: Christopher Davies

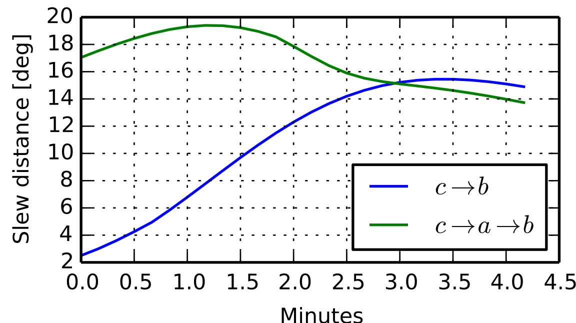

Figure 1: Cumulative slew distance changes over time and violates the triangle inequality after 3 minutes.

Related Work

The Space Mission Planning Problem (Hall and Magazine 1994) seeks to schedule a sequence of jobs that would maximize job satisfaction given time constraints and job priorities. Hall and Magazine assert that the Space Mission Planning Problem is NP-complete by comparison to "Sequencing with release times and deadlines" (Garey and Johnson 1979).

Globus et al. present the Earth Observing Satellite (EOS) Scheduling Problem (Globus et al. 2004) of maximizing observation satisfaction during a fixed period of time along with a fixed set of sensors, constraints, and priorities. The EOS Scheduling Problem and the Space Mission Problem are similar but the EOS Scheduling Problem takes more satellite-specific constraints into consideration. Globus et al. investigate the EOS Scheduling Problem with a genetic approach.

Frank et al. (Frank et al. 2001) examine the EOS Scheduling Problem using a Constraint Based Interval representation of time, and reformulate the problem into a Dynamic Constraint Satisfaction Problem in order to leverage existing algorithms. Frank et al. argue

that this is large and complex enough that a greedy stochastic search method with a resource contention aware heuristic is the best approach.

Aldinger et al. formulate the problem as a numerical planning problem with slew constraints that makes use of consecutive, short planning horizons stitched together by spacecraft orientation at the boundaries. Aldinger et al.'s short planning horizons are a special case of the sliding window replanner described in this paper, with window vertex overlap $m = 0$ and no initial seed schedule.

Aldinger et al.'s windowed approach has good tractability, but restricts optimization to local intervals by scoping both the observation selection and tour scheduling to the planning sub-horizons. This paper seeks to improve Aldinger et al.'s approach by changing the initial conditions of the windowed phase. Instead of starting with an empty schedule, we start with a sub-optimal seed schedule produced by Squeaky Wheel Optimization, which operates globally and converges quickly.

Ichoua et al. explore a related problem in time dependent vehicle routing in which a fleet of vehicles must be scheduled to service multiple customers (Ichoua, Gendreau, and Potvin 2003). The vehicle's travel time varies depending on when it is dispatched. In their approach, they use a parallel tabu search heuristic in order to generate a reasonable solution that is more representative of real-world conditions.

A key tractability assumption in Ichoua et al.'s problem is that the vehicles are undersubscribed, so waiting until a trip becomes cheaper is a nonsensical action that is excluded from the search space (Ichoua, Gendreau, and Potvin 2003). In our problem, however, the single observer is assumed to be oversubscribed with no penalty for deferring a trip. Deferring the trip can actually add value to the schedule, as the observer makes cheap, lower value trips while waiting for the high value trip to become cheaper. The no-wait assumption in Ichoua et al.'s formulation is what makes the time-dependent travel cost vehicle routing problem NP-complete, but the no-wait assumption does not apply to our problem.

Joslin and Clements defined Squeaky Wheel Optimization (SWO) for scheduling (Joslin and Clements 1999). They schedule tasks one at a time, in priority order, at the earliest opportunity. On subsequent iterations, problematic tasks are given higher priorities, the schedule is flushed, and the tasks are scheduled again using the new priorities. The iterations are scored using original task priorities and the best schedule is retained. There is precedent for using Squeaky Wheel Optimization for space observational planning (Knight and Chien 2006; Rabideau et al. 2010; Analytical Graphics, Inc. 2015), though this hybrid approach adds an iterative improvement step.

Pralet and Verfaillie (Pralet and Verfaillie 2014) describe Time-dependent Simple Temporal Networks (TSTNs) for modeling systems where temporal constraints may vary depending on start time. In a TSTN, the temporal distance between two time-points x and y is bounded by functions of x and y. For Earth-observing satellites, the time required for a satellite to slew to and image a target varies with the satellite and target's relative locations, and thus with start times.

Formulation

We restate the Earth Observational Scheduling problem (Globus et al. 2004) as a best effort tour of all vertices with time varying edge costs, cycles within the graph (revisits) and time interval restrictions (visibilities). Given a graph $G = (V, E)$ where each vertex $v_i \in V$ is a desired visit to a science target and time varying transit cost $e_i(t) \in E$ between science target visits, choose the path $P \subseteq G$ that maximizes fitness score $f_{sat} = f(P)$, where a P that visits all V within the planning horizon $[0, t_{end})$ may not exist.

By analogy to the job flow problem of minimizing tardiness with sequence dependent setup costs (Allahverdi et al. 2008), optimal EOS scheduling is expected to be at least NP-hard. We accept sub-optimality and restrict the problem to the looser guarantee of not worse than the prior iteration. The first stage is looped Squeaky Wheel Optimization, with a bound on the maximum number of iterations. The second stage is a looped replan/fill iterative process (figure 2). The replanner is an

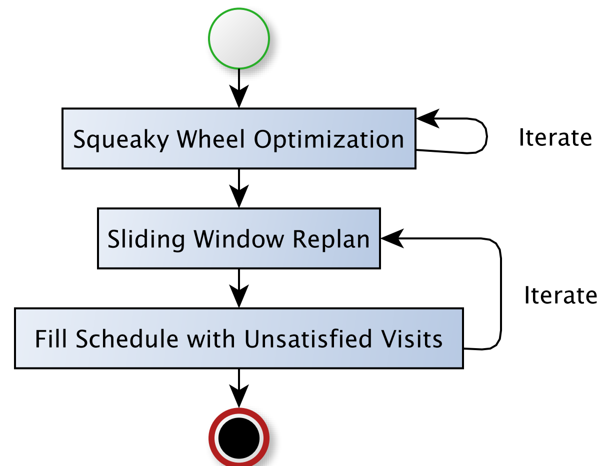

Figure 2: Hybrid Squeaky Wheel Optimization, looped replanning approach

overlapping, sliding window scheme that consolidates visits, creating gaps of idle time in the schedule, but makes no attempt to improve satisfaction score f_{sat}. The fill phase increases f_{sat} by scheduling unsatisfied visits in the gaps, in priority order. The replan/fill loop repeats until all visits are satisfied, replanning/filling produces no meaningful improvement, or a maximum number of iterations is reached. The replan step is locally scoped to a replan window, but the fill step is globally scoped to the entire schedule.

Visits as Temporal Constraint Networks

The primary unit of work for our SWO scheduler is the *Observation Request*. *Observation Requests* provided by an end user specify the targets to be observed, along with observational and operational constraints. These targets can be point locations, geographic areas or trajectories in space (comet/asteroid flybys). Observation requests are fulfilled by *Visits*, which represent the pointing of an instrument at a target which may be revisited. These visits are then fulfilled by actual *Observations* that represent the instrument captures that cover the target, as shown in Figure 3.

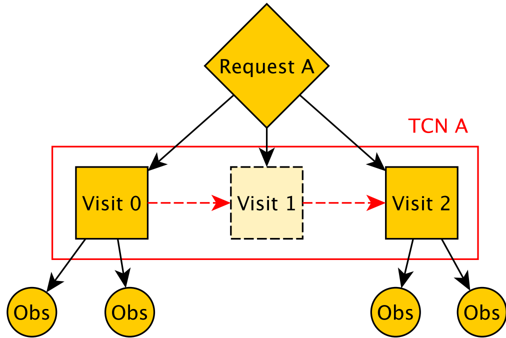

Figure 3: Hierarchy of Observation Requests

We call visits 0 and 2 *detailed* because their child observations have been scheduled on the timeline and all necessary resources (e.g. on-board memory, spacecraft orientation, instrument time) have been reserved. Visit 1 is *abstract* because it has no child observations to commit it to an exact start time. Visit 1 only has eligible intervals defined by the upper and lower bounds of the revisit constraint and its predecessor and successor visits (visits 0 and 2).

The visits of each observation request constitute a Temporal Constraint Network (TCN) (Dechter, Meiri, and Pearl 1991). Each request has its own visit TCN. We apply visit TCNs to this problem as an intermediate layer between an observation request and the observation activities that satisfy the request (red boxes in figures 3, 4). The visits of a given request are a strict temporal network: each visit must start strictly after its predecessor ends, and end strictly before its successor begins. The TCNs of different requests are independent, allowing visits belonging to different requests to be interleaved on the observation schedule (figure 4).

Squeaky Wheel Implementation

Our squeaky wheel implementation iterates over observation requests, scheduling an observation request as an atomic operation. Scheduling an observation request means instantiating the TCN of visits required by the observation, then detailing as much of the TCN as possible given resource availability in the current schedule. We detail each visit as early as possible within its resource availability and revisit constraint window (first

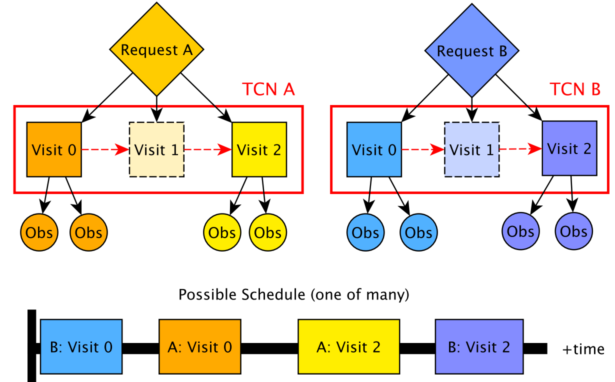

Figure 4: Temporal Constraint Networks are scoped to individual requests. Visits belonging to different requests may be interleaved on the schedule.

fit), hoping to pack the schedule early and leave large gaps later in the timeline for other observation requests. It schedules greedily based on priority (value).

We follow the same basic construct, analyze, prioritize cycle as canonical SWO (Joslin and Clements 1999) but differ in implementation details. Where Joslin and Clements maintain a priority queue between iterations, we maintain two priority values for each request - the true priority (immutable) and the working, adjusted scheduler priority (mutable). In the analyze phase, we do not apply blame based on cost - we apply blame to a request if the scheduler cannot satisfy any part of the request in that iteration. Any failure receives equal blame.

Our priority queue variation is key to our implementation's strict priority scoring and immunity to priority inversion. We retain a mapping of observation request to its current scheduler priority at the start of the iteration. When sorted by scheduler priority, this mapping is a recipe to recreate that iteration's priority queue. We journal this recipe and the overall f_{swo} true-priority score of the resulting schedule. We update the mapping by adding a fixed $\Delta p_{\text{scheduler}}$ to the scheduler priority for each request that received blame in the prior iteration. We do not reset the scheduler priorities between iterations, giving them the same "sticky sort" quality as Joslin and Clements' original implementation (Joslin and Clements 1999).

After all squeaky wheel iterations have been exhausted or all requests have been satisfied, we sort the recipes by their true-priority f_{swo}. We choose the best recipe and follow it to recreate the best schedule discovered by squeaky wheel optimization. Our variation has these traits, most of which are shared with Joslin and Clements' original version:

- A low priority request is attempted before a high priority request if the low priority request is problematic
- Immune to priority inversion in final output

Table 1: Summary of scoring functions

Trait	f_{swo}	f_{sat}	f_{time}	f_{slew}
Rewards value	Yes	Yes		
Priority inversion risk		Yes		
Rewards efficiency			Yes	Yes

- Not trapped at local maxima of schedule score because it does not search the schedule score objective function (Joslin and Clements 1999)
- Not trapped at local minima of schedule cost because it does not search schedule cost
- Can get trapped in a cycle around a specific scheduler priority ordering (Joslin and Clements 1999)
- Biased toward partial satisfaction of requests because only total failure increases scheduler priority
- Not guaranteed to find globally optimal cost or value

Scoring

Successful schedules should maximize satisfaction of observation requests while minimizing non-science time and slews. Our SWO scheduler attempts to maximize satisfaction, while the TSP replanner attempts to minimize time cost. The overall fitness of a schedule is a composite of scores that reward different elements. The scheduling phases have different needs, which we address with different fitness scoring functions. Table 1 summarizes the scoring functions, which are described in more detail below.

Value-based scores Squeaky wheel schedules are scored with strict priority value function f_{swo}, which is invulnerable to priority inversion:

$$p_{\min} = \min\left(p_i, (p_i, r_i) \in R : r_i \text{ satisified}\right) \tag{1}$$

$$f_{swo} = |(p_i, r_i) \in R : r_i \text{ satisified} \wedge p_i = p_{\min}| \tag{2}$$

where R is the set of all Observation Requests attempted and p_i is the priority of request i. Our convention is that smaller p_i means more important/higher priority. In this context, request r_i is satisfied if any of its visits have been detailed.

We compare f_{swo} for two schedules by priority tiers. The schedule with the better p_i in its top tier of satisfied requests wins. If they have the same p_i in their top tier, the schedule with the largest cardinality of satisfied requests at that tier wins. If that is also a tie, we repeat the comparison using the next lower tier on each schedule until all tiers have been exhausted.

Satisfaction score rewards satisfaction of high-priority observation requests. In this context, a request is fully satisfied if its target has been imaged with a sufficient number of revisits. If an area target is only partially imaged, or if insufficient revisits are planned, the request reports partial satisfaction as a fraction of the full goal. If there are r Observation Requests,

$$f_{sat} = \sum_{i=1}^{r} w_i \frac{|v_j \in V_{TCN,i} : v_j \text{ is detailed}|}{|V_{TCN,i}|} \tag{3}$$

where the weight term w_i is a function of request i's priority p_i:

$$w_i = \begin{cases} \frac{1}{p_i}, & 1 \le p_i \\ 2 - p_i, & p_i < 1 \end{cases} \tag{4}$$

Satisfaction score f_{sat} is vulnerable to priority inversion.

Cost-based scores Time cost score f_{time} penalizes idle time between visits.

$$t_{\text{cost}} = \sum \left(t_{start,i+1} - t_{end,i}\right) \tag{5}$$

$$f_{time} = \begin{cases} \frac{1}{t_{\text{cost}}}, & 1 \le t_{\text{cost}} \\ 2 - t_{\text{cost}}, & t_{\text{cost}} < 1 \end{cases} \tag{6}$$

Cost schedule score f_{slew} penalizes large separation angles ϕ between sequential orientation segments that require costly slews.

$$\Phi = \sum |\phi_{i,i+1}| \tag{7}$$

$$f_{slew} = \begin{cases} \frac{1}{\Phi}, & 1 \le \Phi \\ 2 - \Phi, & \Phi < 1 \end{cases} \tag{8}$$

Slew distance f_{slew} and idle time f_{time} scores represent the same concept - cost to complete the tour. They are also both flawed. Idle time score f_{time} penalizes time gaps between visits, but doesn't discriminate between slew time (bad) and idle time (good). f_{slew} has a different problem - it rewards wasting schedule time between successive visits to the same target because the cheapest slew is no slew.

The two cost scores complement the other's weakness, so we use them together. If the window being replanned or the scratchpad timeline it is being replanned on contain more than one visit for the same observation request, we use f_{time} as the cost score. For all other cases, we use f_{slew} as the cost score.

Sliding Window TSP Replanner

We replan by abstracting (lifting) a region of the visit schedule, re-ordering, and rescheduling visits in place according to the new order. We call the function that replans the abstracted visits the *replanning kernel*. Any replanning function can be used but we only detail two in this paper and have only implemented one (the unordered insertion heuristic with time propagation). We expect that replanning kernels will generally have high computational complexity, so we employ them in an overlapping sliding window scheme. This section describes one replanning kernel and how it is used in our sliding window approach.

TSP Insertion Heuristic Replanning Kernel Insertion heuristics are a straightforward approach to producing a satisfactory TSP solution. Starting with a set of vertices and an empty tour, a vertex is removed from the set and inserted into the tour at each possible position; the optimal insertion point is chosen and the process continues until the set is empty. The choice of vertex and insertion point lead to a variety of approaches

outlined in (Reinelt 1994). The lengths of tours generated by insertion heuristics are typically no more than twice that of the optimal tour (Rosenkrantz, Stearns, and Lewis 1977). For our purposes, vertices represent visits, and replanning can be thought of as abstracting previously planned visits and reordering them subject to the TCNs of their parent requests.

We use an unsorted insertion TSP heuristic to replan a subset of the visit schedule produced by squeaky wheel optimization. This meets the general insertion method definition in (Rosenkrantz, Stearns, and Lewis 1977), but differs from the specific implementations in how it selects the next vertex to add to the existing subtour. Insert furthest, nearest, and cheapest all rank the vertices that have not been added by some heuristic function relative to the existing tour (cheapest, closest, or furthest from the tour). Unsorted insertion search chooses an arbitrary vertex to add to the existing tour. We expect unsorted insertion search to produce worse tours in a static cost TSP, but we think it is appropriate here because this problem is a time-varying cost TSP. Past sorting decisions about vertex $i-1$ may be invalidated by an insertion decision made for vertex i.

Begin with a queue of vertices and an empty list of visited vertices. Until the queue is empty, dequeue a vertex and for each possible insertion point, construct a new list with the vertex inserted at that position. From the resulting set of lists, select the one with the least weight as the starting list for the next iteration.

The following theorems prove the runtime complexity of this approach. Theorem 0.1 proves the $\mathcal{O}\left(n^2\right)$ complexity statement in (Rosenkrantz, Stearns, and Lewis 1977) for an unsorted insertion heuristic, laying the foundation for Theorem 0.2. Theorem 0.2 accounts for time-varying slew costs in the TSP to give us the complexity of executing our replanning kernel (unsorted insertion heuristic) over one sliding window.

Theorem 0.1 (Complexity of unsorted insertion search replanning kernel without time propagation)**.** *Given a graph $G(V,E)$ with n vertices, the unsorted insertion heuristic has runtime complexity $\mathcal{O}\left(n^2\right)$.*

Proof. Assume constant time to insert an unvisited vertex into the list and constant time to evaluate the cost of a list. Let $T(i)$ be the number of positions evaluated at the end of the i'th iteration. On the first iteration there is one position to evaluate, thus $T(1) = 1$. On the i'th iteration there will be i positions to evaluate, thus

$$T(i) = \sum_{j=0}^{i} 1 + T(i-1) \tag{9}$$

which is the well known Arithmetic Series, whose solution is $T(n) = n(n+1)/2$ (Cormen et al. 2009). Therefore $\mathcal{O}\left(n^2\right)$. ☐

Example 0.1.1. Let $G(V,E) = K_4$ with vertex labels: $V = \{A, B, C, D\}$ and edge weights:

$$d(i,j) = E = \begin{pmatrix} \infty & 1 & 2 & 3 \\ 1 & \infty & 1 & 2 \\ 2 & 1 & \infty & 1 \\ 3 & 2 & 1 & \infty \end{pmatrix}$$

Let π represent a sequence of vertices and $c(\pi) = \sum_{i=1}^{n-1} d(\pi_{i-1}, \pi_i)$. Let Π_i be the set of candidate branches after iteration i with $\Pi_0 = \{A\}$. Select $\pi_i^* = \arg\min_{\pi\in\Pi_i} c(\pi)$ as the optimal starting point for iteration $i+1$. Select B from V, then $c(AB) = 1$ $c(BA) = 1$ yielding $\pi_1^* = AB$. Draw C from V, then $c(CAB) = 3$, $c(ACB) = 3$, $c(ABC) = 2$ yielding $\pi_2^* = ABC$. Draw D from V, then $c(DABC) = 5$, $c(ADBC) = 6$, $c(ABDC) = 4$, $c(ABCD) = 3$ yielding $\pi_3^* = ABCD$. Now $V = \emptyset$, and report $\pi^* = \pi_3^*$, which for this worked example is the global optimal tour π_g^*.

Theorem 0.2 (Complexity of unsorted insertion search replanning kernel with time propagation)**.** *Given a graph $G(V,E)$ with n vertices, the insertion heuristic has runtime complexity $\mathcal{O}\left(n^3\right)$ if start times are committed for each vertex $v \in V$.*

Proof. Derived from the time propagation consequence in the time-dependent travel time TSP (Ichoua, Gendreau, and Potvin 2003). Because start time of any vertex $v_j \in V_i$ at iteration i has been committed, checking insertion point j requires that start time of v_j and all following vertices be recomputed.

Replacing the constant time 1 insertion cost term in equation 9 with $i-j$ vertices after position j,

$$\begin{aligned}
T(i) =& \sum_{j=0}^{i} (i-j) + T(i-1) \\
=& \sum_{j=0}^{i} i - \sum_{j=0}^{i} j + T(i-1) \\
=& \frac{1}{2}\left(i^2 - i\right) + T(i-1) \\
=& \frac{1}{2}\left(i^2 - i + (i-1)^2 - (i-1)\dots\right) \\
=& \frac{1}{2}\left(\sum_{j=0}^{i} (i-j)^2 - \sum_{j=0}^{i} j\right) \\
=& \frac{1}{2}\left(i^3 - 2\sum_{j=0}^{i} ij + \sum_{j=0}^{i} j^2\right) + \frac{i\,(i+1)}{2} \\
\therefore T(n) =& \mathcal{O}\left(n^3\right)
\end{aligned}$$

☐

Theorem 0.2 is important for our sliding window replanner for two reasons. First, we require that each sub-solution remain feasible, including the time allocated for slews between requests. Slew time is a function of time, so we must at least temporarily commit the visits to a start time. The other reason is that all

feasible solutions of a replan window will have the same f_{sat}, so f_{slew} and f_{time} break ties[1]. These require slew computations, and therefore time commitments.

Sliding Window The $\mathcal{O}\left(n^3\right)$ complexity of the unsorted insertion search heuristic with time propagation replanning kernel would be intractable for large schedules. To make large schedules tractable, we restrict the $\mathcal{O}\left(n^3\right)$ portion to a small n using a sliding window scheme.

Begin with a queue of unvisited vertices, and an empty list of visited vertices. Until the queue is empty, dequeue n unvisited vertices and pass them to the kernel in the same order. The last m elements of the kernel's output are cleared from the kernel's output and placed at the beginning of queue in the same order, and the remaining $n-m$ elements placed at the end of the visited sequence.

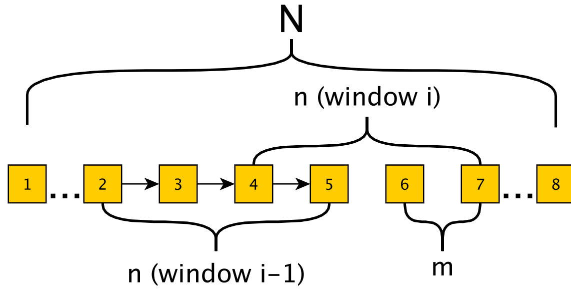

Figure 5: Sliding Window

Theorem 0.3 (Complexity of replanning an entire schedule with a sliding replan window)**.** *Given N vertices, and replanning kernel complexity f, the sliding window produces $\mathcal{O}\left(\left\lceil \frac{N}{n-m} \right\rceil f(n)\right)$ runtime complexity.*

This theorem aggregates complexity of the unsorted insertion search heuristic with time propagation sliding window replanning kernel (theorem 0.2) over an entire schedule. Figure 5 shows how the terms in this theorem relate to the schedule being replanned.

Proof. Every iteration n dequeues, $\mathcal{O}(f(n))$ operations, m deletions from the kernel's output, m insertions into the queue, and $n-m$ insertions into the list are performed. Thus, $T(i) = 2n + m + f(n)$. There are $\left\lceil \frac{N}{n-m} \right\rceil$ iterations in total. Thus, T(N) $= (2n+m+f(n)) \left\lceil \frac{N}{n-m} \right\rceil$. Therefore, $\mathcal{O}\left(\left\lceil \frac{N}{n-m} \right\rceil f(n)\right)$ □

Corollary 0.3.1. *Assuming that $f \in \mathcal{O}\left(n^d\right)$ for $d \in \mathbb{N}$, $n = g(N) \in o(N)$ and $m \ll n$, yields complexity $\mathcal{O}\left(Ng(N)^{d-1}\right)$. For example, if $g(N) = \ln N$, then $\mathcal{O}\left(N \ln{(N)}^{d-1}\right)$ is obtained.*

Example 0.3.1. Let $G(V,E) = K_8$ with vertex labels: $V = \{A, B, C, D, A', B', C', D'\}$ and edge weights given by:

$$d(i,j) = E = \begin{pmatrix} \infty & 1 & 2 & 3 & 10 & 1 & 2 & 3 \\ 1 & \infty & 1 & 2 & 1 & 10 & 1 & 2 \\ 2 & 1 & \infty & 1 & 2 & 1 & 10 & 1 \\ 3 & 2 & 1 & \infty & 3 & 2 & 1 & 10 \\ \infty & 1 & 2 & 3 & \infty & 1 & 2 & 3 \\ 1 & \infty & 1 & 2 & 1 & \infty & 1 & 2 \\ 2 & 1 & \infty & 1 & 2 & 1 & \infty & 1 \\ 3 & 2 & 1 & \infty & 3 & 2 & 1 & \infty \end{pmatrix}$$

Let $n = 8$, $m = 4$, and the kernel $\kappa : \Pi \times E \to \Pi$ be the insertion heuristic. Let $\pi_0 = CADBC'A'D'B'$, then $\pi_0' = \kappa(\pi_0, E) = DCBC'B'AD'A'$. The first $n-m$ vertices are placed on $\pi_w = DCBC'$, and remaining m on $\pi_1 = B'AD'A'$. The next iteration $\pi_1' = \kappa(\pi_1, E) = D'B'AA'$, and repeating the allocation yields $\pi_w = DCBC'D'B'AA'$ and $\pi_2 = \emptyset$ and we terminate.

Methodology

Two experiments are performed with this hybrid approach. First, a toy problem demonstrates a suboptimal schedule generated by strict priority SWO alone, and the impact of replanning on this suboptimal schedule. The second experiment evaluates the performance of the replanning phase against a random target deck with randomized revisit constraints.

Toy Problem Experiment

Observation requests are created in a circle around a point on the observer's ground track. Request priorities are specified such that SWO produces a suboptimal path with many edge crossings. Revisit intervals are hand-selected such that every request may be visited before any request is due for revisit.

As this is a toy problem, the experiment will be evaluated qualitatively. The algorithm will be successful if the number of edge crossings is reduced, with a desired outcome of no crossings and a circular path between the requests.

Squeaky wheel scheduling is expected to schedule all visits of all *observation requests* in a fragmented schedule with no large gaps (saturated activity schedule). The sliding window TSP replanning algorithm is expected to re-order visits such that the path is approximately circular, with shorter slews, little fragmentation and one large gap per visit cycle. Because the replanning algorithm operates locally on partial schedules, we expect a sub-optimal solution (not a circle). The replanned schedule will be judged qualitatively by the number of edge crossings it removes from the path and

[1] See Formulation:Scoring:Cost-based scores for more information on when f_{slew} is used and when f_{time} is used.

how much more condensed the schedule is after replanning.

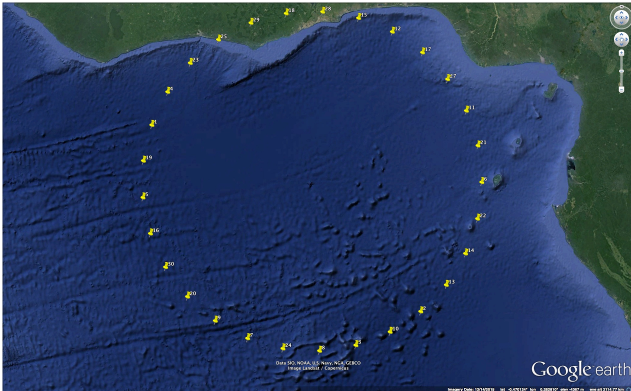

Figure 6: Targets aligned in a ring

Random Points/Visits Experiment

The purpose of this experiment is to evaluate the impact of replanning dense, more realistic schedules. The SWO schedule is treated as a baseline (iteration 0). Both the quality improvement (improvement in schedule score) and runtime cost-effectiveness (improvement in schedule quality for runtime) of replanning are examined.

Observation requests are created for randomly generated points with random revisit intervals. The experiment is repeated with three different agility profiles (high, medium and low agility). If sliding window replanning is effective, we expect significant improvements in schedule score each iteration.

Results

Toy Problem

As expected, the initial Squeaky Wheel Optimization schedule produced a long tour with edge crossings (figure 7). After one pass of sliding window replanning, most of the edge crossings are removed (figure 8). After a second replanning pass (not shown), all edge crossings were removed. Figure 8 highlights how drastically even one pass of replanning improves tour efficiency.

Figure 9 shows that after the squeaky wheel optimization but before replanning (OrientationTimeLine_swo), the schedule is fragmented. After one pass of replanning (OrientationTimeLine_hybrid), the visits are clustered near the start of the timeline, leaving the latter two-thirds of the timeline free for other science requests. Figure 9's large gap after replanning suggests that overall schedule fitness score f_{sat} could be improved by a looped, multi-pass replan/fill phase.

Random Points/Visits

Table 2 shows that sliding window replan/fill iterations had the greatest impact for the lowest computational cost for the low agility observer's schedule. This is likely because replanning focuses on reducing slew costs, and slews are more costly for a low agility observer than a high agility observer.

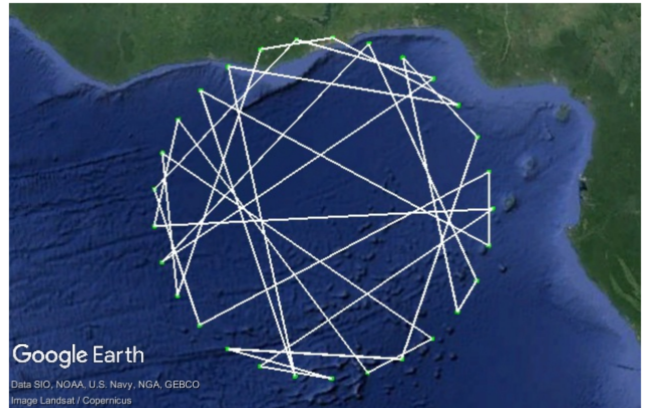

Figure 7: Schedule produced by Squeaky Wheel Optimization. Note edge crossings.

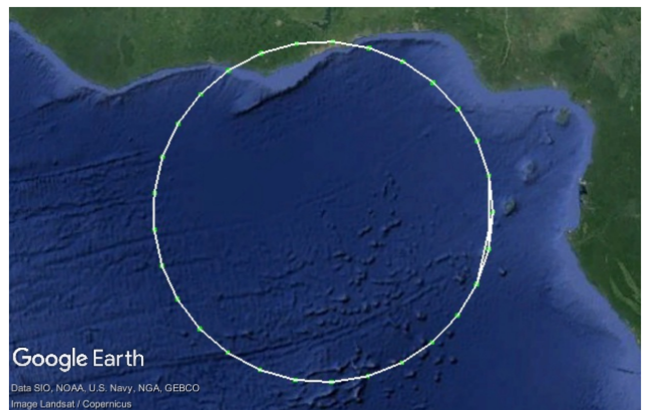

Figure 8: Schedule after sliding window TSP replanning. Note reduction in edge crossings.

Table 2: Total replanning score improvement

Agility	Runtime	Δf_{sat}	Visits Added
High	9.1 min	0.0016	1.4 %
Med	6.3 min	0.0037	3.0 %
Low	5.4 min	0.0080	7.7 %

Figure 10 shows that all agility models receive some satisfaction score increase due to looped replan/fill loops. Satisfaction score f_{sat} converges asymptotically on an upper bound for all three agility profiles. Because we schedule in priority order, early iterations schedule the high-priority requests that contribute the most to f_{sat}, leaving only low priority requests for later fill iterations. Later iterations mostly minimize slew cost and idle time, with little satisfaction score f_{sat} im-

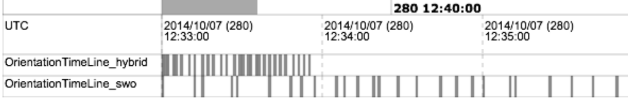

Figure 9: Comparison of timelines before replanning (SWO) and after replanning (hybrid).

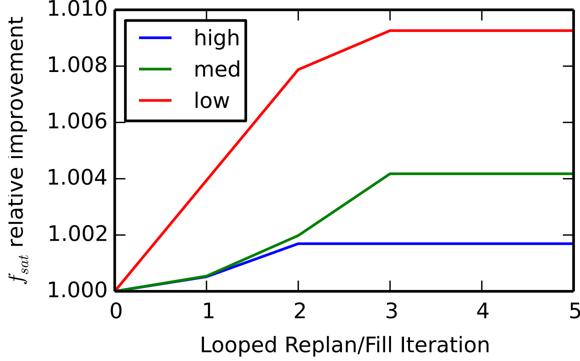

Figure 10: Relative score (as fraction of starting score) by looped replan/fill iteration. Note score convergence.

provement. After iteration four, replanning produces no meaningful satisfaction score improvement (figure 11). Another interpretation is that the actions of the replan/fill loop primarily address starvation of low priority requests without resorting to dueness or tardiness terms in the value function.

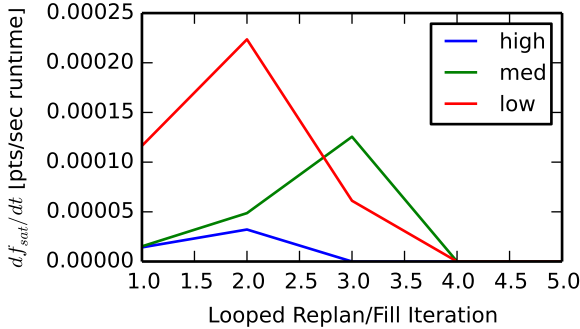

Figure 11: Score improvement return on computational runtime cost.

Discussion

While implementing this hybrid algorithm, we discovered several quirks and edge cases of this scheduling problem. We discuss them below.

TSP Replanner is Sensitive to Input Order

A consequence of time varying slew costs is that early decisions about vertex sequencing can become invalidated by subsequently added vertices. The cheaper transition may be $b \to a$ when $V = \{a, b\}$, but adding vertex c to V may cause $a \to b$ to become cheaper. All vertex c must do is perturb time and delay the slew between a and b (figure 12).

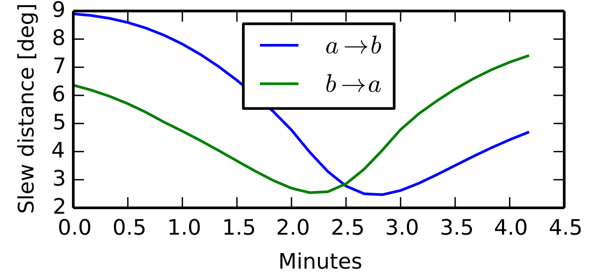

Figure 12: Slew distance between a and b can be both asymmetric and time varying.

As a consequence, we bias towards local features which may neglect the global characteristics we want to deliver in planning. We do not backtrack and reconsider sequencing decisions made when the graph was small, so we cannot even claim that the path within a replanned window is optimal.

Replanning May Fail to Maintain f_{sat}

The Temporal Constraint Network revisit constraints are used as an additional visibility constraint. With larger schedules, it is possible for replanning in an earlier window to cause a subsequent visit in its TCN to become due before it enters a replan window. If enough other visits occurred between the dueness window start and when the replan window slides contains the visit, the revisit can come too late.

This violates our fundamental goal of not reducing the score of a schedule during iterated replanning. We compromise by accepting the violation of the revisit constraint for this iteration and retain the SWO solution for this replan window, then advance the window. The hope is that a future replan window may alter the overall schedule in a way that the replan window covers this visit when it is in its dueness window.

Complexity Control: Scratchpads

Our initial implementation of the replanner copied the entire timeline (N vertices) for each graph variation in each sliding window. This altered the complexity of the window's kernel f in theorem 0.3 from $f(n)$ to $Nf(n)$. When substituted into theorem 0.3 and our insertion heuristic kernel f,

$$\mathcal{O}\left(\left\lceil \frac{N}{n-m} \right\rceil f(n)\right)$$

became

$$\mathcal{O}\left(\left\lceil \frac{N^2}{n-m} \right\rceil f(n)\right) \approx \mathcal{O}\left(N^2\right)$$

We sought to avoid quadratic complexity in our replanner. Linear complexity was achieved with *scratch-*

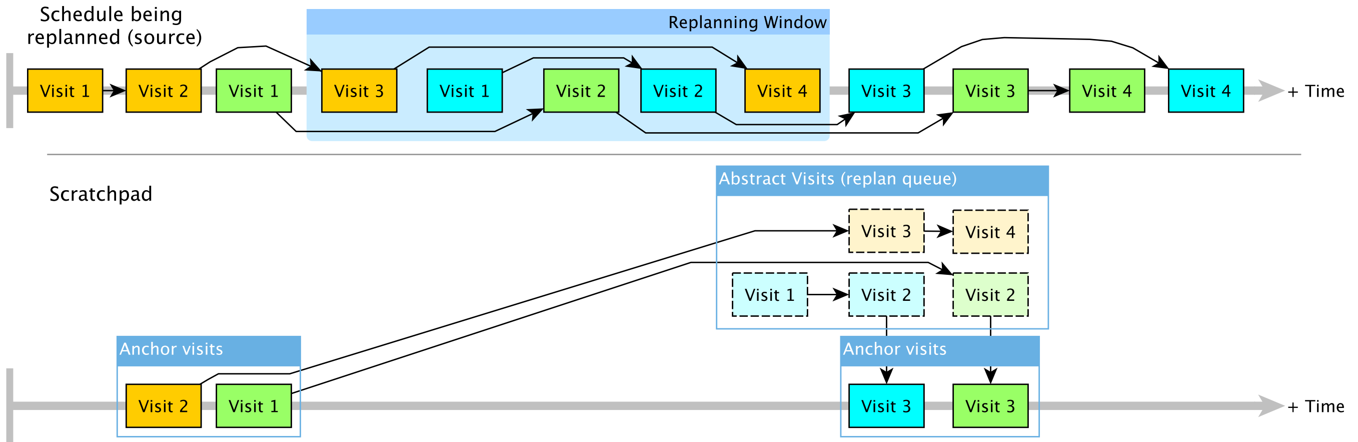

Figure 13: Scratchpad as a sparse subset of the scheduled being replanned. Green visit 1 and blue visit 3 are bookends.

pads: partial timelines copied to the sliding window replanner.

Scratchpads avoid unnecessary duplication of the working timeline when calling an inner tour planner. A scratchpad contains visits to be placed in a particular window of time, bookended by the visits immediately before and after the visits to be scheduled. The bookend visits serve to enforce a slew feasibility constraint - events that are replanned must start as a slew from the leading bookend and end with a feasible slew to the tailing bookend. We perform all of our hypothetical scheduling on scratchpads, then transfer the visits from the best scratchpad back to the working timeline.

We must also represent the memory commitments outside of the replan window - just before and after the scratchpad bookend visits. We replace all memory fills and drains outside of the replan window with two equivalent bookend fill reservations. These bookends are propagation limits that also detect when a memory reservation inside the replan window would invalidate a reservation after the replan window. The leading bookend is a ramp (constant rate) from 0 to the recorder fill level at the start of the replan window. The tailing memory bookend is a ramp from the replanning window end to the highest fill level after the replan window, but before a communications access completely drains the data recorder. Memory bookend reservations repropagate in constant time.

Additionally, we inform the replanner of the revisit constraints that apply to the visits within the replanner. We do this by gathering the visit TCNs affected by the replan window, then copying the first detailed visit prior to the replan window and the first detailed visit after the replan window to the scratchpad as detailed *Anchor Visits*. The anchors constrain the replanner's search complexity to only the sub-TCNs within the replan window.

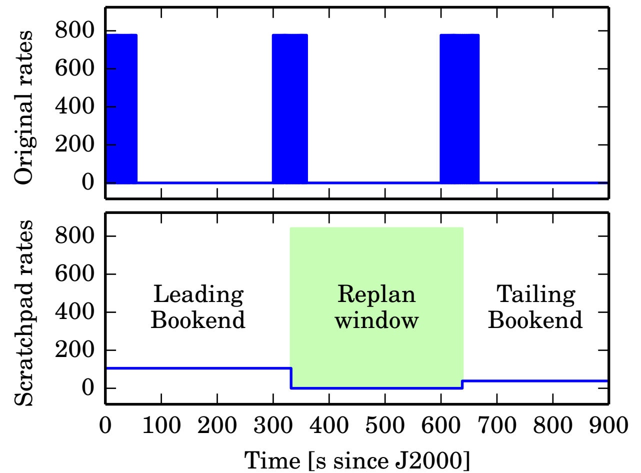

Figure 14: Memory scratchpad with simplifying bookend rate reservations

Future Work

The initial ordering of visits could be generated by minimum spanning trees, convex hulls, sweeping lines, or other methods, instead of SWO. As long as the initial schedule satisfies all (or as many as possible) requests, replanning should generate an acceptable schedule.

Greedy heuristics that minimize edge cost at each step tend to produce poor asymmetric TSP solutions, while greedy heuristics that minimize tolerances find much better solutions. (Goldengorin and Jäger 2005) Instead of the currently-used insertion heuristic, the replanner could use a tolerance-based greedy approach, or other alternative TSP heuristics.

Another approach for the replanning phase could maintain the sliding window to reduce the runtime complexity, and incrementally improve the tour using existing approaches such as pair wise exchange (Croes 1958),

tabu search, and simulated annealing.

The time score f_{time} could be improved. As the goal of the replanner is to improve efficiency and create gaps that can accommodate unsatisfied requests/visits, f_{time} should reward idleness instead of penalizing it. Idleness could be computed by subtracting the minimum slew duration between two consecutive orientation segments (artifacts of visit detailing) from the duration of the gap between the two orientation segments.

Conclusions

The sliding window TSP-based replanner improved the quality of the schedule for all three observer agility profiles examined in the experiment. Squeaky Wheel Optimization was responsible for the vast majority of the schedule value score improvement. Looped replanning/filling primarily reduced starvation of lower priority requests that would otherwise not be scheduled by Squeaky Wheel Optimization. Looped replanning/filling had a more significant impact on the schedule of a low agility observer than it did for higher agility observers. Future work should focus on improving the quality of the replan phase and testing replan scores that reward schedule defragmentation (idleness).

Acknowledgments

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

The authors would like to thank the reviewers for their constructive feedback. It significantly improved this paper.

References

Aldinger, J.; Lohr, J.; Winker, S.; and Willich, G. 2013. Automated planning for earth observation spacecraft under attitude dynamical constrants. In *Jahrbuch der Deutschen Gesellschaft für Luft- und Raumfahrt*.

Allahverdi, A.; Ng, C.; Cheng, T. E.; and Kovalyov, M. Y. 2008. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research* 187(3):985–1032.

Analytical Graphics, Inc. 2015. Algorithm definitions. Web. Accessed: 2017-03-17.

Chien, S.; Rabideau, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F.; Barrett, T.; et al. 2000. Aspen–automated planning and scheduling for space mission operations. In *International Conference on Space Operations (SpaceOps 2000)*, 1–10.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2009. *Introduction to Algorithms, 3rd Edition:*. MIT Press.

Croes, G. 1958. A method for solving traveling salesman problems. *Operations Research* 6:791–812.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial intelligence* 49(1-3):61–95.

Frank, J.; Jonsson, A.; Morris, R.; and Smith, D. 2001. Planning and scheduling for fleets of earth observing satellites. In *In Proceedings of Sixth International Symposium on Artificial Intelligence, Robotics, Automation and Space*.

Garey, M., and Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman.

Globus, A.; Crawford, J.; Lohn, J.; and Pryor, A. 2004. A comparison of techniques for scheduling earth observing satellites. In *Proceedings of the 16th conference on Innovative Applications of Artificial Intelligence*. AAAI Press.

Goldengorin, B., and Jäger, G. 2005. How to make a greedy heuristic for the asymmetric traveling salesman problem competitive. Technical report, University Groningen.

Hall, N., and Magazine, M. 1994. Maximizing the value of a space mission. *European journal of operational research* 78:224–241.

Ichoua, S.; Gendreau, M.; and Potvin, J.-Y. 2003. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research* 144:379–396.

Joslin, D. E., and Clements, D. P. 1999. Squeaky wheel optimization. *Journal of Artificial Intelligence Research* 10:353–373.

Karger, D.; Motwani, R.; and Ramkumar, G. D. 1997. On approximating the longest path in a graph. *Algorithmica* 18(1):82–98.

Knight, R., and Chien, S. 2006. Producing large observation campaigns using compressed problem representations. In *Proceedings of the 8th International Workshop of Planning and Scheduling for Space (IWPSS-2006)*.

Knight, R.; Donnellan, A.; and Green, J. J. 2013. Mission design evaluation using automated planning for high resolution imaging of dynamic surface processes from the iss. In *Proceedings of the 8th International Workshop of Planning and Scheduling for Space (IWPSS-2013)*. Jet Propulsion Laboratory, National Aeronautics and Space Administration.

Lemaître, M.; Verfaillie, G.; Jouhaud, F.; Lachiver, J.-M.; and Bataille, N. 2002. Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology* 6:367–381.

Pinedo, M. L. 2012. *Scheduling: Theory, Algorithms, and Systems*. Springer Science+Business Media.

Pralet, C., and Verfaillie, G. 2014. Time-dependent simple temporal networks: Properties and algorithms. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*.

Rabideau, G.; Chien, S.; McClaren, D.; Knight, R.; Anwar, S.; Mehall, G.; and Christensen, P. 2010. A tool

for scheduling themis observations. In *In Proceedings of Sixth International Symposium on Artificial Intelligence, Robotics, Automation and Space*.

Reinelt, G. 1994. *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag.

Rosenkrantz, D. J.; Stearns, R. E.; and Lewis, II, P. M. 1977. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing* 6(3):563–581.