# Optimal Fixed Interval Satellite Range Scheduling

Antonio J. Vazquez [1] and R. Scott Erwin[2]
[1]*National Research Council Post-Doctoral Fellow, Albuquerque, NM, U.S.A.*
[2]*Principal Research Aerospace Engineer, Air Force Research Laboratory, Albuquerque, NM, U.S.A.*

Keywords: Scheduling Algorithm, Polynomial Time Algorithm, Earth Observation Satellite, Ground Station Network.

Abstract: The satellite scheduling community has provided several algorithms for allocating interaction windows between ground stations and satellites, from simple greedy approaches to more complex hybrid-genetic or Lagrangian-relaxation techniques. Single-location ground station problems, where requests have fixed time intervals and no priorities, are known to be solvable in polynomial time. To the best of our knowledge, no algorithm has been provided yet for solving multiple-location, prioritized scheduling problems optimally. We present an exact polynomial time algorithm for a fixed number of ground stations (or satellites), based on a modified algorithm from the general scheduling literature.



## 1 INTRODUCTION

The problem of scheduling interactions between ground stations and satellites is widely-known in the literature as *Satellite Range Scheduling* (SRS). Whereas ground stations can be considered to be moving with the surface of the Earth, satellites travel through different kinds of orbits. These two different motion dynamics generate visibility time windows when lines of sight (LOS) between satellites and ground stations exist.

The interactions are limited to occur within these visibility windows, and could also involve time constraints specified by the operators of the satellites, which depending on the specific mission may require fixed or variable contact times lasting for the entire window duration or just a portion of it.

In this paper we focus on the case where these windows are defined by fixed times, so for the sake of simplicity we assume that the passes are generated directly from the LOS times among ground stations and satellites. That is generally the case for Low Earth Orbits (LEOs), due to their short times (L. Barbulescu, 2004) (for more insight on these models see (Vallado, 2001)); and it is possible to decompose time-discrete variable-intervals problems into fixed-interval scheduling problems (Wolfe and Sorensen, 2000).

Even for the fixed interval case, the problem of optimally scheduling satellite-to-ground station contacts, in terms of meeting operator requirements and optimizing the amount of data exchanged, rapidly increases in complexity as the number of entities increases.

There are two main fields of application on satellite scheduling, ground station networks (GSN) and Earth observation satellites (EOS) (Burrowbridge, 1999; Wolfe and Sorensen, 2000; H. Jung, 2002; L. Barbulescu, 2004; F. Marinelli, 2005). Despite the differences in the applications, which are basically related to the actions taken during the contact times, both problems can be approached through the same mathematical model, which we present in this paper. For general problems on fixed-interval scheduling see (A. W. J. Kolen, ; M. Y. Kovalyov, 2007).

In this paper we provide the first algorithm which solves the fixed interval SRS problem in polynomial time for a fixed number of ground stations (or satellites). The algorithm is based on one from general scheduling from ref. (Arkin and Silverberg, 1987), which has been modified to fit SRS constraints, and improved for a faster execution.

The paper is organized as follows. In Section 2 we briefly present the formulation of the fixed SRS problem, which allows us to describe the algorithm and its differences with its reference in Section 3. In Section 4 we present a detailed example on the development of the algorithm, and two simulation examples to shed some light on the influence of the scenario in the performance of the algorithm. In Section 5 we present the conclusions of the research and further lines of application for the algorithm.

J. Vazquez A. and Scott Erwin R..
Optimal Fixed Interval Satellite Range Scheduling.
DOI: 10.5220/0004760604010408
In *Proceedings of the 3rd International Conference on Operations Research and Enterprise Systems* (ICORES-2014), pages 401-408
ISBN: 978-989-758-017-8

## 2 MATHEMATICAL MODEL FOR THE FIXED INTERVAL SRS PROBLEM

Let $S = \{s_h\}$ be a set of satellites, $G = \{g_i\}$ a set of ground stations, $t_0$ a time instant and $T$ a time window such that $t \in [t_0, t_0 + T]$. The literature establishes a classification on SRS depending on the number of entities: *single resource range scheduling problem (SiRRSP)* considers that either the number of satellites $|S| = 1$ or the number of ground stations $|G| = 1$, whereas *multiple resource range scheduling problem (MuRRSP)* is the general case where both $|S| \geqslant 1$ and $|G| \geqslant 1$. For the sake of generality we consider the multiple resource case in the remaining development.

**Definition 1.** Let a *pass* $p_l$ be a tuple modeling a visibility time window from a start time $t_s$ to an end time $t_e$ between the satellite $s_h$ and the ground station $g_i$, and with an assigned priority $w_l$, that is

$$p_l = (s_h, g_i, t_{s_l}, t_{e_l}, w_l) \ : \ w_l \in \mathbb{R} \cap [0,1]. \quad (2.1)$$

The term $w_l$ characterizes the weight or *priority* associated to that request, normalized between 0 and 1. Let $P = \{p_l\}$ be a set of $|P| = N$ passes.

**Definition 2.** Every subset $P_{\text{sub}} \subseteq P$ is a *schedule* (a set of passes that might be tracked). Note that a schedule defined this way is not necessarily feasible, in the sense that it may require the same resource for two concurrent passes.

Let $c_\Sigma$ be a conflict indicator function which yields a 1 if two passes $p_u, p_v \in P_{\text{sub}} : u, v \in \mathbb{N} \cap [1, N]$ are overlapping in time (condition $[c_2]$ in eq. 2.2) for a single ground station or satellite ($[c_1]$ in eq. 2.2) and they are generated by different requests ($[c_3]$ in eq. 2.2). For the sake of clarity, for each $p_l \in P_{\text{sub}} \subseteq P$, the functions $g$ and $s$ are defined, so that $g(p_l) = g_i \Leftrightarrow p_l = (s_h, g_i, t_{s_l}, t_{e_l}, w_l)$, and the same for $s(p_l)$ and $s_h$) for accessing the applicable elements in the tuple. We define $c_\Sigma$ as follows:

$$c_\Sigma : p_u, p_v \longrightarrow \begin{cases} 1, & \text{if } [c_1] \cap [c_2] \cap [c_3], \\ 0, & \text{otherwise,} \end{cases}$$
$$[c_1] = \{(g(p_u) = g(p_v)) \cup (s(p_u) = s(p_v))\}, \quad (2.2)$$
$$[c_2] = \{(t_{s_v} \in [t_{s_u}, t_{e_u}]) \cup (t_{e_v} \in [t_{s_u}, t_{e_u}])\},$$
$$[c_3] = \{u \neq v\}.$$

Let $C_\Sigma$ be the analogous function to be applied to a schedule:

$$C_\Sigma : P_{\text{sub}} \longrightarrow \sum c_\Sigma(p_u, p_v) \ \ \forall p_u, p_v \in P_{\text{sub}}. \quad (2.3)$$

Furthermore no *preemption* is allowed, which means that passes must complete once started before switching to another pass; and there are no *precedence* relations between the passes, that is, no pass is a pre-requisite for antother pass.

**Definition 3.** A schedule $P_{\text{sub}}$ is considered *feasible* if it has no conflicts:

$$P_{\text{sub}} \in \{P_\Sigma^{\text{f}}\} \Leftrightarrow C_\Sigma(P_{\text{sub}}) = 0, \quad (2.4)$$

where $\{P_\Sigma^{\text{f}}\}$ is the space of all the feasible schedules.

The main objective in SRS problems is to maximize the sum of the weights $w_l$ of the passes in the feasible schedule. Given a schedule $P_{\text{sub}}$, let the metric $\|\cdot\|_{\Sigma w}$ be:

$$\|P_{\text{sub}}\|_{\Sigma w} = \sum_l w_l \ : \ w_l = w(p_l) \ \forall p_l \in P_{\text{sub}}. \quad (2.5)$$

**Definition 4.** The problem of Satellite Range Scheduling can be stated as finding an *optimal schedule* $P^*$, which is a feasible schedule with maximal metric:

$$P^* \in \{P_\Sigma^{\text{f}}\}, \ \nexists P_{\text{sub}} \in \{P_\Sigma^{\text{f}}\} : \ \|P_{\text{sub}}\|_{\Sigma w} > \|P^*\|_{\Sigma w}. \quad (2.6)$$

## 3 OPTIMAL SOLUTION FOR THE FIXED INTERVAL SRS PROBLEM

In this section we present an algorithm strongly based on the one in ref. (Arkin and Silverberg, 1987) (Theorem 3), and provide the main contributions of the paper.

This referenced algorithm generates a graph representing all the legal states of the system, where nodes are vectors modeling the state of all the machines, and edges represent the legal transitions among these nodes. However this algorithm can not be directly applied in the SRS problem since it would not take into account conflicts on one satellite being tracked by two different ground stations.

The main contributions of the paper are in the modifications in the extension of the nodes with new events (expression 3.5) for avoiding the existence of unfeasible states in the diagram, and the dynamic calculation of the longest path during the graph creation through the avoidance of certain edges (expression 3.7) and the propagation of the weights of the passes during the graph creation (both expressions 3.5 and 3.7).

This new constraint on the feasible states avoids both conflicts in ground stations and satellites, as in the reference only conflicts of a kind are detected, thus providing an optimal solution for the SRS problem.

The second contribution improves the performance of the algorithm, also benefited from the fact that passes are associated to only one ground station and one satellite.

To the best of our knowledge, no algorithm has provided an optimal solution in polynomial time to this specific problem before.

## 3.1 Description of the Algorithm

The algorithm can be broken down into three main steps:

- *Event generation*, where passes are converted into events, each event defining the stages in the generated graph.
- *Graph creation*, where nodes and edges are progressively created following the list of events, so that two kind of stages exist depending on whether the event is associated to an *start time* or to an *end time* of a pass.
- *Shortest path calculation*, which provides the set of nodes with maximal metric, and thus the optimal schedule. As it will be detailed later, the way the graph is created allows for a dynamic calculation of the path.

In the following sub-sections we describe formally these steps. For a detailed example of the algorithm see subsection 4.1.

### 3.1.1 Event Generation

Similarly to the algorithm in ref. (Arkin and Silverberg, 1987), passes are mapped into events $e = \{t, \phi, s, g, w_x\}$, defined by their start and end times ($t \in \{t_s(p_l), t_e(p_l)\}$), a sign ($\phi \in \{+1, -1\}$) regarding if it is a start or an end time, a ground station $g \in \{1, 2, ..., k_2\}$, a satellite $s \in \{0, 1, 2, ..., k_1\}$ (the element 0 applies when the ground station is idle), and a priority $w_x$. Without loss of generality and to keep the notation compact, ground stations are assumed to be the scheduling resources (or machines in general scheduling problems), although they can be interchanged. The two bijective functions $f_v^+(p_l) = e^+$ and $f_v^-(p_l)) = e^-$ are defined for separating between start time $e^+$ and end time $e^-$ events:

$$e^+ = (t_s(p_l), +1, s(p_l), g(p_l), w(p_l)), \tag{3.1}$$

$$e^- = (t_e(p_l), -1, s(p_l), g(p_l), 0). \tag{3.2}$$

Applying the functions $f_v^+$ and $f_v^-$ to a set of passes $P$ yields the two sets of events $E^+$ and $E^-$ respectively. Let $E$ be the set of $2N$ events generated from $P$, sorted by ascending time.

$$E = \langle \{e_i\} \rangle \ : \ e_{i-1} \prec e_i \Leftrightarrow t(e_{i-1}) < t(e_i), \quad e_i \in E^+ \cup E^-. \tag{3.3}$$

### 3.1.2 Graph Creation

The graph generation will be performed in stages (or layers, as in ref. (Arkin and Silverberg, 1987)). Every event $e_i$ will be associated to an stage $Z_i$, and these stages can be seen as sets of nodes. The nodes represent the status of all the ground stations (scheduling entities), so each node $n_j$ will be a vector with their associated satellites.

$$n_j = (s(n_j, g_1), s(n_j, g_2), ..., s(n_j, g_{k_2})) \ : \ s(n_j, g_i) \in \{0, 1, 2, ..., k_1\}, \tag{3.4}$$

where $s(n_j, g_i)$ is the satellite assigned to the $i$-th ground station in node $n_j$.

Let the *frontier* $B_{i-1}$ be the set of nodes which are checked for modification or deletion at any stage $Z_i$ during the graph generation.

In the *first stage* $Z_0$ there is only one node, wherein all the resources are empty $\exists^* n_0 \in Z_0; |Z_0| = 1; n_0 = (0, 0, ..., 0)$, the frontier includes only this node $B_0 = \{n_0\}$, and for the consistency of the algorithm the previous frontier is empty $B_{-1} = \emptyset$ and the edge from $n_0$ to $\emptyset$ has null weight $\exists^* v = (n_0, \emptyset, 0)$.

For the stage $Z_i$, nodes $n$ and edges $v$ are generated based on the nodes of the frontier $B_{i-1}$ associated to the previous stage $Z_{i-1}$, and on the event $e_i$ associated to the current one $Z_i$.

**Start Time Event Stages.** Nodes are generated from those in the frontier wherein the ground station $g$ indicated in the event is idle, so that new nodes keep their state unmodified but the resource $g(e_i)$, which takes its value ($s(e_i)$) from the event.

$$\begin{aligned} &\forall n_j \in B_{i-1} : s(n_j, g(e_i)) = 0, \\ &\qquad s(n_j, g') \neq s(e_i) \ \forall g', \\ &\qquad \exists^* v' = (n_j, n_x, w_x), \ n_x \in B_{i-2}, \\ &\text{if } \ \phi(e_i) > 0, \text{ then} \\ &\exists^* n_l \in Z_i : s(n_l, g(e_i)) = s(e_i), \\ &\qquad s(n_l, g') = s(n_j, g') \ \forall g' : \\ &\qquad g' \neq g(e_i), \\ &\exists^* v = (n_l, n_j, w_x + w(e_i)). \end{aligned} \tag{3.5}$$

The frontier of the new stage includes all the nodes of the frontier from previous stage plus all the nodes in the new stage.

$$B_i = B_{i-1} \cup Z_i \tag{3.6}$$

Note the new condition on $s(n_j, g')$ in expression 3.5 contrasting to the referenced algorithm, as this last one does not consider this kind of conflicts among passes (or *jobs* in general scheduling literature). Note also the introduction of the propagation of the weights $w_x + w(e_i)$ in the creation of the edges $v$.

**End Time Event Stages.** In these stages, one node is created for each pair of nodes with all the resources keeping the same state except the one indicated in the event, and one edge is created from the one of the pair which has the highest accumulated weight in its path to this new node.

$$\begin{aligned}&\forall n_j \in B_{i-1} : s(n_j, g(e_i)) = s(e_i),\\ &\qquad\qquad \exists^* v' = (n_j, n_x, w_j),\ n_x \in B_{i-2},\end{aligned}$$

and if $\phi(e_i) < 0$, then also

$$\begin{aligned}&\exists^* n_y \in B_{i-1} : s(n_y, g(e_i)) = 0,\\ &\qquad\qquad s(n_y, g') = s(n_j, g'),\ \forall g' :\\ &\qquad\qquad g' \neq g(e_i),\\ &\exists^* v'' = (n_y, n_z, w_y),\ n_z \in B_{i-2},\end{aligned} \tag{3.7}$$

hence

$$\exists^* n_l \in Z_i : n_l = n_y,$$

$$\exists^* v = \begin{cases} (n_l, n_j, w_j), & \text{if } w_j \geqslant w_y,\\ (n_l, n_y, w_y), & \text{if } w_j < w_y.\end{cases}$$

At the end of the stage the new nodes are added to the frontier, and the evaluated ones deleted. Let $A_i(l) = \{n_l\}$ and $D_i(l) = \{n_j, n_y\}$ be the added and deleted sets from each new node $n_l$ at stage $Z_i$, then the frontier $B_i$ can be expressed as follows:

$$B_i = \left\{ B_{i-1} \cup \bigcup_l A_i(l) \right\} - \bigcup_l D_i(l). \tag{3.8}$$

Note the selection of the edge with the highest weight $w_j$ or $w_y$ in expression 3.7, which dismisses sub-optimal paths, contrasting to the referenced algorithm which postpones the calculation of the shortest path after generating the whole graph. In this sense note also the propagation of this highest weight to the new edge $v$.

### 3.1.3 Shortest Path Calculation

All the edges are directed from one stage to the previous, no node is duplicated at start event stages, and only one edge of the two which would merge is created at end event stages; so that the generated graph is a tree with edges connecting nodes from the leaves towards the root (which is the start node $n_0$).

Backtracking a path from any of the leaves in the tree is trivial due to the unitary oudegree of all the nodes in the graph:

$$\deg^+(n_j) = 1 \ \ \forall j \geqslant 0, \tag{3.9}$$

Thus, the longest path is obtained in the backtracking process starting at the last created node.

## 3.2 Optimality of the Solution and Complexity of the Algorithm

**Theorem 1.** *The SRS problem with fixed number of ground stations or satellites and a set of passes $P = \{p_1, p_2, ..., p_N\}$ with associated weights $w_i$ and fixed times $(t_{s_i}, t_{e_i})$ $\forall p_i \in P$ can be solved in $O(N(k_1+1)^{k_2})$, where $k_1$ is the number of satellites or ground stations and (the fixed number) $k_2$ is the complementary, and $N$ is the number of passes.*

**Proof.** We will prove optimality of the solution by revision of the properties of the generated graph, and polynomial time sovability by accounting for all the necessary steps of the algorithm for the worst case.

Suppose we relax the node creation in expression 3.7 to create both edges (to $n_y$ and $n_j$). By definition, according to the algorithm description (subsection 3.1.2), adding more nodes or edges to the graph would bring unfeasible states or transitions, and deleting nodes or edges would remove feasible states or transitions. Now, if we delete this relaxation, we will delete unoptimal subpaths but keep the optimal one.

Since this graph is directed (edges are oriented) and acyclic ($\exists v = (n_x, n_y, w) \Rightarrow n_x \in Z_i\ ; n_y \in Z_{i-1};\ w > 0$), and there are both an start and an end nodes, a longest path algorithm can be applied to obtain the optimal set of states in polynomial time (Arkin and Silverberg, 1987).

This set of nodes in the longest path can be easily transformed into the set of associated passes. By definition (subsection 3.1.1), there is a bijection between the passes in $P$ and the events in $E^+$. Let $Z^+$ be the subset of all the stages $Z$ associated to events in $E^+$ (also related by a bijective association). Given that $\exists v = (n_x, n_y, w) \Rightarrow n_x \in Z_i\ ; n_y \in Z_{i-1}$ and also $n_j \in Z_x, Z_y \Leftrightarrow Z_x = Z_y$, there is an biunivocal association between the subset of nodes in $Z^+$ in any path and the subset of passes which generated the associated stages. Therefore this is applicable to the path between the end and the start nodes, completing the proof of optimality.

The worst case for the number of nodes in a frontier occurs when all the satellites $k_1 = |S|$ can be assigned to all the ground stations $k_2 = |G|$. Considering also the idle state, an upper bound for the number of nodes is $\max\{|B_i|\} = (k_1+1)^{k_2}$ (subsection 3.1.1). Since there are $2N$ stages (actually $2N+1$, but first and last nodes can be grouped together inside an only "worst case frontier"), an upper bound to the number of states checked during the graph creation is:

$$\max\left\{ \sum_{i=1}^{2N} |B_i| \right\} < 2N(k_1+1)^{k_2}. \tag{3.10}$$

Even if the ordering of the $2N$ events is taken into account, existing algorithms can provide an ordered set $E$ in $O(2N\log(2N))$ (Musser, 1997). Even the easiest non-trivial problem $k_1 = 2,\ k_2 = 1$ allows for a relatively high number of events holding the inequality $\log(2N) < (k_1+1)^{k_2}$, so that we can dismiss the complexity of the ordering for practical problems.

Furthermore, since all the edges are directed from stage $Z_i$ to stage $Z_{i-1}$, and from eq. 3.9 all the nodes have only one edge, backtracking the longest path is trivial starting from the end node (subsection 3.1.3). Note that given that the nodes keep the weight of the path to the initial node, the latest generated node has the highest value. Since the maximum number of nodes to backtrack is $2N$ (as many as stages), the process can be done in $O(N)$.

Thus the algorithm runs in $O(N(k_1+1)^{k_2})$. □

It is possible that the generated directed acyclic graph (DAG) has frontiers with an only node, wherein all the resources are idle. In this case the DAG can be split into several subgraphs (separated by these frontiers), which can be solved independently. This allows to provide results earlier by serialization or parallelization.

# 4 APPLICATION EXAMPLES

We provide an example detailing the creation of the graph and calculation of the optimal schedule, and two simulations for shedding some light on the influence of the scenario in the performance of the algorithm.

## 4.1 Graph Generation

This example aims to explain the proposed algorithm in a MuRRSP simple scenario entailing ground stations $g_1$ and $g_2$, satellites $s_1$ and $s_2$, and 4 passes (one for each pair station - satellite). The pass intervals are represented in Figure 1.

The list of the four passes is $P = \{p_1, p_2, p_3, p_4\}$ which we extend into the set of events $\{e_1, e_2, ..., e_8\}$:

$$p_1 = (s_1, g_1, t_1, t_4, w_1) \begin{cases} e_1 = (t_1, +1, s_1, g_1, w_1), \\ e_2 = (t_4, -1, s_1, g_1, 0), \end{cases}$$
$$p_2 = (s_2, g_1, t_3, t_6, w_2) \begin{cases} e_3 = (t_3, +1, s_2, g_1, w_2), \\ e_4 = (t_6, -1, s_2, g_1, 0) \end{cases}$$
$$p_3 = (s_1, g_2, t_2, t_7, w_3) \begin{cases} e_5 = (t_2, +1, s_1, g_2, w_3), \\ e_6 = (t_7, -1, s_1, g_2, 0), \end{cases}$$
$$p_4 = (s_2, g_2, t_5, t_8, w_4) \begin{cases} e_7 = (t_5, +1, s_2, g_2, w_4), \\ e_8 = (t_8, -1, s_2, g_2, 0), \end{cases}$$

where $w_1 = 0.6$, $w_2 = 0.6$, $w_3 = 0.8$ and $w_4 = 0.4$.

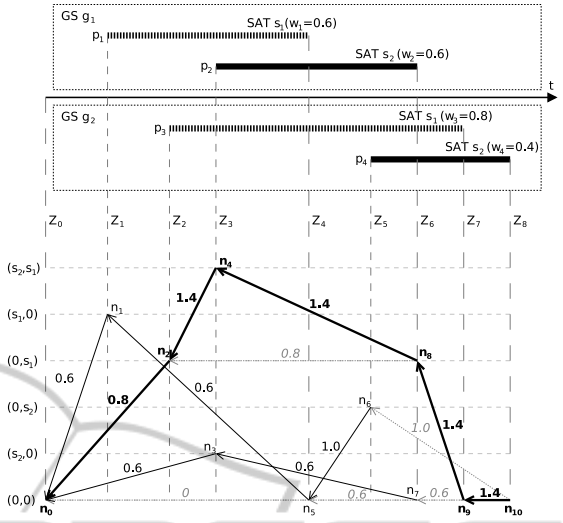


Figure 1: Graph generation for the Example 1.

We obtain the set $E$ by sorting the events by increasing time, so that:

$$E = \{e_1, e_5, e_3, e_2, e_7, e_4, e_6, e_8\}. \quad (4.1)$$

**Stage $Z_0$:** We initialize the algorithm with $i = 0, B_{-1} = \emptyset, n_0 = (0,0), v_0 = (n_0, \emptyset, 0)$ and $Z_0 = B_0 = \{n_0\}$.

**Stage $Z_1$:** We take the first element in $E$, which is $e_1$. There is only one node to evaluate in the frontier $B_0$, which is $n_0$. This node complies with the conditions in first paragraph of eq. 3.5: *(a)* there is no satellite assigned to the ground station indicated in the event ($s(n_0, g(e_1)) = s(n_0, g_1) = 0$), *(b)* the satellite indicated in the event is not assigned to any other ground station ($s(n_0, g') \neq s(e_1) = s_1 \forall g'$), and *(c)* we have the edge $v_0 = (n_0, \emptyset, 0)$, so that we proceed to create the new node $n_1$.

From the second paragraph of expression 3.5, the new node has the same values for all its elements except for the ground station indicated in the event $g(e_1) = g_1$, which had assigned a zero, and now has assigned $s(e_1) = s_1$, so that $n_1 = (s_1, 0)$. We also create the new edge from the new node ($n_1$) to the examined one ($n_0$), with a weight equal to the sum of the edge from $n_0$ to $\emptyset$ and the weight of the event $w(e_1) = w_1$, so that $v_1 = (n_1, n_0, w_1)$.

There are no more nodes to examine in $B_0$, so that $Z_1 = \{n_1\}$, and $B_1 = B_0 \cup Z_1 = \{n_0, n_1\}$.

**Stage $Z_2$:** The next event in $E$ is $e_5$, which is also a start time event. In this case we evaluate the nodes in $B_1$, which are $n_0$ and $n_1$. Note that $n_1$ can not be extended, because $s(e_5) = s(n_1, g_1)$, i.e. the satellite indicated in the event is already assigned to another ground station. Note that this avoids the conflict were

two ground stations would be tracking the same satellite.

Then we can only extend node $n_0$, which following the procedure in previous stage yields a new node $n_2 = (0, s_1)$, and an edge $v_2 = (n_2, n_0, w_3)$. Then, $Z_2 = \{n_2\}$, and $B_2 = B_1 \cup Z_2 = \{n_0, n_1, n_2\}$.

**Stage $Z_3$:** Also following previous procedure, we examine $n_0$, $n_1$ and $n_2$. Node $n_1$ is dismissed since $g(e_3) = g_1$ has already assigned $s_1$ ($s(n_1, g(e_3)) = s_1 \neq 0$).

Extending nodes $n_0$ and $n_2$ we create the pairs $n_3 = (s_2, 0), v_3 = (n_3, n_0, w_2)$ and $n_4 = (s_2, s_1), v_4 = (n_4, n_2, w_3 + w_2)$, so that $Z_3 = \{n_3, n_4\}$ and $B_3 = \{n_0, n_1, n_2, n_3, n_4\}$.

**Stage $Z_4$:** Next event in $E$ is $e_4$, which is an end time event ($\phi(e_4) < 0$). According to the conditions in expression 3.7, we only examine the nodes in $B_3$ which have the satellite $s(e_4) = s_2$ assigned to the ground station $g(e_4) = g_1$, which is only the node $n_1 = (s_1, 0)$. Note that $v_1 = (n_1, n_0, w_1)$, and also there is an only node in $B_3$ which has the same state that $n_1$ except for the only change in the ground station $g(e_4) = g_1$, which has assigned a zero, and this node is $n_0 = (0, 0)$, with $v_0 = (n_0, \emptyset, 0)$.

We thus will select one of the two nodes, $n_0$ or $n_1$, the one with the highest accumulated weight on its associated vector ($v_0$ or $v_1$). Since the weight of $v_0$ is lesser than that of $v_1$ we select $n_1$ as the end of the edge to be created from the new node. We create then the new node $n_5 = (0, 0)$ and the new edge $v_5 = (n_5, n_1, w_1)$. Note that this selection dismisses unoptimal sub-paths.

As no more nodes from $B_3$ can be extended, the set of created nodes is $\{n_5\}$, and the set of deleted nodes is $\{n_0, n_1\}$. The new frontier is $\{B_3 \cup \{n_5\}\} - \{n_0, n_1\}$, which is $B_4 = \{n_2, n_3, n_4, n_5\}$.

**Rest of Stages:** Following the same procedures with the rest of the events we generate the rest of nodes and edges,

$$
\begin{array}{ll}
n_0 = (0,0), & v_0 = (n_0, \emptyset, 0), \\
n_1 = (s_1,0), & v_1 = (n_1, n_0, 0.6), \\
n_2 = (0,s_1), & v_2 = (n_2, n_0, 0.8), \\
n_3 = (s_2,0), & v_3 = (n_3, n_0, 0.6), \\
n_4 = (s_2,s_1), & v_4 = (n_4, n_2, 1.4), \\
n_5 = (0,0), & v_5 = (n_5, n_1, 0.6), \\
n_6 = (0,s_2), & v_6 = (n_6, n_5, 1.0), \\
n_7 = (0,0), & v_7 = (n_7, n_3, 0.6), \\
n_8 = (0,s_1), & v_8 = (n_8, n_4, 1.4), \\
n_9 = (0,0), & v_9 = (n_9, n_8, 1.4), \\
n_{10} = (0,0), & v_{10} = (n_{10}, n_9, 1.4),
\end{array}
$$

and the stages and frontiers:

$$
\begin{array}{ll}
Z_0 = \{n_0\}, & B_0 = \{n_0\}, \\
Z_1 = \{n_1\}, & B_1 = \{n_0, n_1\}, \\
Z_2 = \{n_2\}, & B_2 = \{n_0, n_1, n_2\}, \\
Z_3 = \{n_3, n_4\}, & B_3 = \{n_0, n_1, n_2, n_3, n_4\}, \\
Z_4 = \{n_5\}, & B_4 = \{n_2, n_3, n_4, n_5\}, \\
Z_5 = \{n_6\}, & B_5 = \{n_2, n_3, n_4, n_5, n_6\}, \\
Z_6 = \{n_7, n_8\}, & B_6 = \{n_6, n_7, n_8\}, \\
Z_7 = \{n_9\}, & B_7 = \{n_6, n_9\}, \\
Z_8 = \{n_{10}\}, & B_8 = \{n_{10}\}.
\end{array}
$$

We calculate the longest path walking the graph from the last node: $n_{10}, n_9, n_8, n_4, n_2, n_0$. From this set, only nodes $n_2$ and $n_4$ correspond to start time events, and specifically those from passes $p_3$ and $p_2$.

We show the generated graph in Figure 1, where we mark the longest path in bold lines. Those edges which have not been created (but checked for creation) have been represented as light grey dotted lines.

## 4.2 Simulation: Practical Case

For this example a scenario consisting of several ground stations and LEO satellites has been considered, where the visibility windows of the satellites over the ground stations generate the passes. The algorithms have been implemented in MATLAB, and the simulations were run on a virtual machine with a 3 GHz processor and 2 GB RAM.

The greedy algorithm is known to be optimal for the FI-SiRRSP with no prioritization among passes (Burrowbridge, 1999; L. Barbulescu, 2004). This algorithm selects, given a list of passes ordered by non-descending end time, the next pass which can be tracked, and mark as unfeasible all the passes in the list which are conflicting with it. Its worst case complexity is $O(N^2)$.

However the greedy algorithm is not generally optimal for the MuRRSP (nor for the SiRRSP with priorities). A simple example is provided in an scenario with 3 passes, all mutually conflicting but second with third. In this case the first one would be selected, whereas the optimal decision would have been selecting second and third ones. For the prioritized case, consider a scenario with only two conflicting passes where the second has a higher priority than the first one.

The selection of the heuristic is crucial for the performance of the greedy algorithm. Apart from the end-time ordering heuristic introduced in (Burrowbridge, 1999), we also considered for the simulations the ordering based on the priorities of the passes (and end-times for passes with equal priority). Note that neither this heuristic provides an optimal schedule in the FI-SiRRSP with priorities, consider a sce-

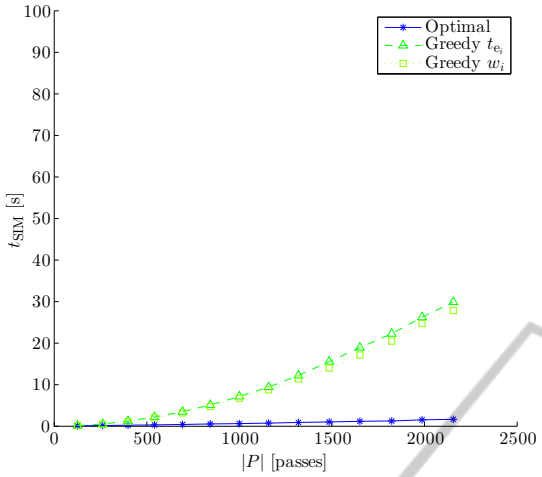


Figure 2: Simulation times for the Example 2.

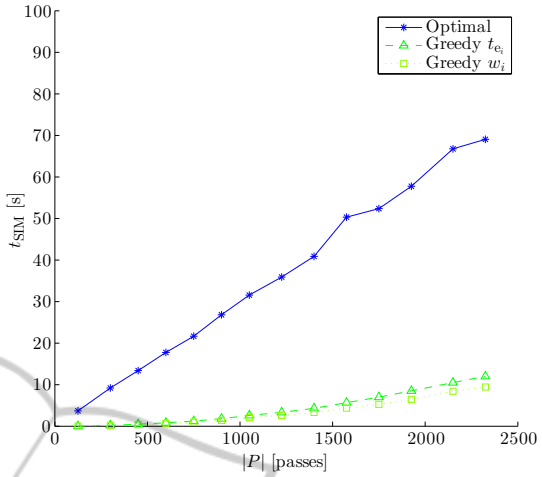


Figure 4: Simulation times for the Example 3.

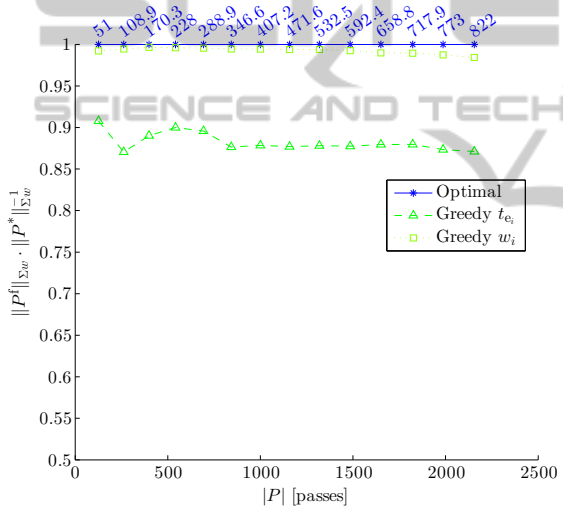


Figure 3: Metric ratios for the Example 2.

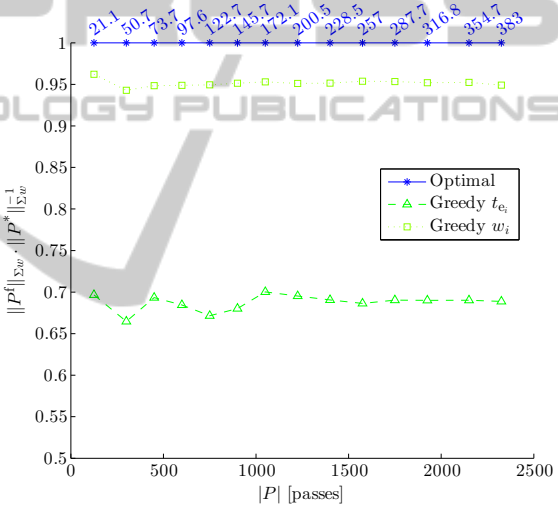


Figure 5: Metric ratios for the Example 3.

nario with three passes where first conflicts with second, second with third, and the priority of the second is greater than others' but smaller than other's sum.

We run the three algorithms on a scenario with $|G| = 5$ stations and $|S| = 5$ satellites, varying the size of the scheduling window from 1 to 14 days. We only use these numbers to show how the dynamics of the problem influence the complexity of the algorithm. As the number of entities increases, better times are clearly foreseeable for the greedy algorithm based on the complexity bounds of the algorithms. Furthermore, passes are given a random priority $w_i = v_i/10 : v_i \sim \mathrm{U}[1,10]$.

Figure 2 shows the execution time of the three algorithms versus the number of passes $|P|$ (corresponding to the scheduling windows for $T = \{1,2,...,14\}$ days. It can be seen that the execution time of the optimal algorithm grows linearly with the number of passes, whereas the greedy algorithms' do quadratically. Increasing the number of entities will imply higher execution times for the optimal algorithm, however still outperforming the times of the other algorithms for high $|P|$ values.

Figure 3 shows the ratio of the metrics (defined in expression 2.5) $\|P^f\|_{\Sigma w} \cdot \|P^*\|^{-1}_{\Sigma w}$, for which the value of the metric $\|P^*\|_{\Sigma w}$ of the optimal algorithm is displayed. Note that the priority-ordering heuristic achieves near-optimal schedules, but never optimal, for the studied cases.

## 4.3 Simulation: Worst Case

This example corresponds to the worst case scenario for the presented algorithm, wherein all the locations

are identical, and so are all the orbits; so that this case yields the maximum number of conflicts. As for the previous case, priorities are generated randomly. Of course this does not correspond to a practical scenario, but will allow to benchmark the worst case behaviour of the algorithm.

Simulation times are displayed in Fig. 4, with similar considerations as those given for Fig. 2. Note that whereas the linear coefficient for the execution time of the optimal algorithm grows, the greedy algorithm reduces its execution time. However, as the scheduling window extends, the execution time for the greedy-based algorithms get worse.

Although we provided upper bounds for the complexity of the presented algorithm, it will be relaxed as the number of conflicts diminishes, or equivalently, with the dispersion of the locations and orbits. This can be easily concluded from the way the graph is constructed, taking into account all the possible combinations of tracked passes (see Figs. 2 and 4). The greedy algorithm improves however its performance as the number of conflicts rise, as for every pass selection a group of passes can be dismissed, leading to a shorter execution time. We conjecture that the increase in the number of conflicts reduces the likelihood of the heuristic algorithms to find a near-optimal solution (see Figs. 3 and 5).

It is worth reminding that the schedules given by the greedy algorithm would be optimal for this example if the priorities were all the same.

## 5 CONCLUSIONS

In this paper we have provided the first exact algorithm in polynomial time for the Fixed Interval SRS problem with a fixed number of ground stations or satellites, based on the algorithm presented in ref. (Arkin and Silverberg, 1987) for general scheduling.

Even though the presented algorithm runs in polynomial time, the tractability can be compromised for cases where the number of ground stations and satellites is high, and locations and orbits are respectively highly correlated. In this sense approaches toward online scheduling (Papadimitriou and Yannakakis, 1989) should be pursued.

These results for fixed interval scheduling can be extended to more general cases, through the discretization of the variable size requests. We are working on this generalization, as well of in distributed scheduling approaches.

## ACKNOWLEDGEMENTS

This research was performed while the author held a National Research Council Research Associateship Award at the Air Force Research Laboratory (AFRL).

We thank Configurable Space Microsystems Innovations & Applications Center (COSMIAC, www.cosmiac.org) for the infrastructure support during this research.

We thank six anonymous reviewers for their valuable comments for improving the manuscript.

## REFERENCES

A. W. J. Kolen, J. K. Lenstra, C. H. P. F. C. R. S. Interval scheduling: A survey.

Arkin, E. M. and Silverberg, E. B. (1987). Scheduling jobs with fixed start and end times. In *Discrete Applied Mathematics, Vol. 18, pp. 1-8*. North-Holland.

Burrowbridge, S. E. (1999). Optimal allocation of satellite network resources. In *Master Thesis*. Virginia Polytechnic and State University.

F. Marinelli, F. Rossi, S. N. S. S. (2005). A lagrangian heuristic for satellite range scheduling with resource constraints. In *Computers & Operations Research, Vol. 38, Issue 11, pp. 1572-1583*. Elsevier.

H. Jung, M. Tambe, A. B. B. C. (2002). Enabling efficient conflict resolution in multiple spacecraft missions via dcsp. In *Proceedings of the NASA workshop on planning and scheduling*. NASA.

L. Barbulescu, J. P. Watson, L. D. W. A. E. H. (2004). Scheduling space-ground communications for the air force satellite control network. In *Journal of Scheduling, Vol. 7, Issue 1, pp. 7-34*. Kluwer Academic Publishers.

M. Y. Kovalyov, C. T. Ng, T. C. E. (2007). Fixed interval scheduling: Models, applications, computational complexity and algorithms. In *European Journal of Operations Research, Vol. 178, pp. 331-342*. Elsevier.

Musser, D. R. (1997). Introspective sorting and selection algorithms. In *Software: Practice and Experience, Vol. 27, Issue 8, pp. 983-993*. Wiley.

Papadimitriou, C. H. and Yannakakis, M. (1989). Shortest paths without a map. In *Lecture Notes in Computer Science, Vol. 372, pp. 610-620*. Springer.

Vallado, D. A. (2001). Fundamentals of astrodynamics and applications. Space Technology Library.

Wolfe, W. J. and Sorensen, S. E. (2000). Three scheduling algorithms applied to the earth observing systems domain. In *Management Sience, Vol. 46, No. 1, pp. 148-168*. Informs.