



An Optimizing Space Data-Communications Scheduling Method and Algorithm with Interference Mitigation, Generalized for a Broad Class of Optimization Problems

James Rash

NASA Goddard Space Flight Center, Greenbelt, Maryland

The NASA STI Program Office ... in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- **TECHNICAL PUBLICATION.** Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.
- **TECHNICAL MEMORANDUM.** Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- **CONTRACTOR REPORT.** Scientific and technical findings by NASA-sponsored contractors and grantees.

- **CONFERENCE PUBLICATION.** Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.
- **SPECIAL PUBLICATION.** Scientific, technical, or historical information from NASA programs, projects, and mission, often concerned with subjects having substantial public interest.
- **TECHNICAL TRANSLATION.** English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at <http://www.sti.nasa.gov/STI-homepage.html>
- E-mail your question via the Internet to help@sti.nasa.gov
- Fax your question to the NASA Access Help Desk at (443) 757-5803
- Telephone the NASA Access Help Desk at (443) 757-5802
- Write to:
NASA Access Help Desk
NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076

Available from

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161

Available electronically at <http://gltrs.grc.nasa.gov>.



An Optimizing Space Data-Communications Scheduling Method and Algorithm with Interference Mitigation, Generalized for a Broad Class of Optimization Problems

James Rash

NASA Goddard Space Flight Center, Greenbelt, Maryland

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

James Rash revised the original NASA Technical Memorandum after his retirement, during his term in the NASA Emeritus Program. He can be contacted through the Office of Human Capital Management, NASA's Goddard Space Flight Center, Greenbelt, Maryland 20771, USA.

Acknowledgments

The author thanks Frank J. Stocklin of NASA Goddard for supporting this research, which culminated in both the original draft of the algorithm and its embodiment in a prototype automated interference-mitigation scheduler. David Joesting of Allied Signal Technical Services Corporation is thanked for useful criticisms on an early draft of the algorithm. Allen J. Levine of NASA Goddard (retired) is thanked for generating data from the Network Planning and Analysis System (NPAS) for comparison with output from the implementation of the predecessor of the herein disclosed algorithm, and is thanked for useful comments on a draft of this paper.

This document was prepared with open-source software for the Macintosh® computer: the LATEX application program TeXShop and the bibliographic application program BibDesk. Also used was the Macintosh built-in application Grapher. The author wishes to acknowledge the developers for the quality of these applications.

Document Change History

This document, numbered as NASA-TM-2014-215855_Rev.1, replaces the previous version, NASA-TM-2010-215855. Changes include the following:

1. Expanded index
 2. Added reference
 3. Errata
 - a. Corrections of mathematical expressions
 - b. Typographical corrections and minor corrections/alterations (punctuation, word-choice, and spelling corrections, formatting changes, etc.), silently incorporated
 4. Definitions added for completeness
 5. Explanatory remarks added
 6. Passages or remarks added or sentences reworded for clarity
 7. Mathematical expressions simplified or revised for clarity
-

Notice for Copyrighted Information

This manuscript has been authored by *James Rash* with the National Aeronautics and Space Administration. The United States Government has a non-exclusive, irrevocable, worldwide license to prepare derivative works, publish, or reproduce this manuscript, and allow others to do so, for United States Government purposes. Any publisher accepting this manuscript for publication acknowledges that the United States Government retains such a license in any published form of this manuscript. All other rights are retained by the copyright owner.

Trade names and trademarks are used in this report for identification only. Their usage does not constitute an official endorsement, either expressed or implied, by the National Aeronautics and Space Administration.

Level of Review: This material has been technically reviewed by technical management

Contents

1	Introduction	7
1.1	Background on NASA Data-Communications Scheduling	7
1.2	Toward an Optimizing Scheduler	7
1.3	Incorporating RF Interference-Mitigation Constraints	7
1.4	Concerning Processes, Methods, and Algorithms	8
1.5	The Essential Technology Used in the Present Disclosure	8
1.6	Innovations Disclosed	9
1.7	Audience for the Present Disclosure	9
1.8	Organization of Paper	9
2	Space-Data Communications-Scheduling Problem Definition	10
2.1	Space-Data Communications Scheduling	10
2.1.1	Scheduling-System Objective	10
2.1.2	Primary Factors Affecting Achievability of Objective	10
2.1.3	Technical Factors Affecting Achievability of Objective	10
2.2	Size of the Solution Space	10
2.2.1	Determining Factors	10
2.2.2	A Simple Example	11
2.2.3	A More Realistic Example	11
2.2.4	Possible Approach to Reducing the Size of the Solution Space	12
2.3	Alternative Approaches	13
2.3.1	Brute Force and Constructive Techniques	13
2.3.2	Necessity of Special Search Techniques	13
2.4	Evolutionary Search	13
2.4.1	Genetic Algorithms	13
2.4.2	Representations of Solutions of the Scheduling Problem	13
2.4.3	Processes for Selection and Creation of Working Population Members	14
2.4.4	Principle of Operation of Genetic Algorithms	14
2.4.5	Progress of Evolutionary Search	14
2.4.6	Effect of Internal Parameters of the Genetic Algorithm	15
2.4.7	Fitness Functions, Minima, Maxima, and Optima	15
2.4.8	Metrics for Evaluation of the Scheduling System	16
2.4.9	Random Numbers and Their Role	16
3	Communications-Scheduling Assumptions	16
3.1	Necessary Input Data	16
3.2	Scope Limitations	17
3.2.1	Two-Week Scheduling on a Weekly Cycle	17
3.2.2	Dynamic Rescheduling	17
3.2.3	Near-Earth Communications Environment	17
4	Definitions	17
4.1	Basic Space-Data-Communications Definitions	17
4.2	General Notation	18
4.3	Domain-Specific Definitions	21
4.3.1	System Input Data	21
4.3.2	Scheduling-Algorithm-Specific Definitions	24
5	Optimal Schedule-Generation Algorithm	35
5.1	Specification of Optimal Schedule-Generation Algorithm	35
5.2	Schedule-Generation Algorithm: Internal Parameters	36

6	Internal-Parameter Optimization: The $S^\#$ Problem	37
6.1	The $S^\#$ Problem: Introduction	37
6.2	The $S^\#$ Problem: Definitions	38
6.3	Algorithm for Solving the $S^\#$ Problem	38
6.4	The $S^\#$ Problem: Discussion	39
7	A Further Abstraction: The $S^{\#\#}$ Problem	40
7.1	The $S^{\#\#}$ Problem: Introduction	40
7.2	Regression-Analysis Approach	40
7.3	Operational Use of Derived Estimation Function	41
7.4	The $S^{\#\#}$ Problem: Discussion	41
8	Implementation	42
8.1	Prototype Implementation of Predecessor Algorithm	42
8.2	Implementing the Disclosed Methods and Algorithms	42
9	Conclusion	43
10	Appendix A. Algorithm Performance	44
10.1	Best-Solution Fitness (Function of Algorithm-Iteration Count)	44
10.2	Assumed-Model Versus Actual Performance	44
10.3	Fitness as a Function of a Single Parameter	46
10.4	Fitness as a Function of Elapsed Run Time	48
10.5	Key Insight Afforded by the Assumed Model	49
11	Appendix B. The Generalized Algorithms and Methods	49
11.1	The General Problem	49
11.1.1	Introduction	49
11.1.2	Basic Definitions	49
11.1.3	Associating Generalized Problem Domains with Real Problem Domains	50
11.1.4	A Note on Applicability	50
11.1.5	The Essential Evolutionary Search Functions	51
11.2	The Type-G Problem	53
11.3	Type-G Problems: The G-Algorithm	53
11.3.1	Execution of a G-algorithm	54
11.4	The $G^\#$ Problem	54
11.4.1	Regarding Type-G Meta Problems	54
11.4.2	Meta-Problem Scenarios Compounded Indefinitely	54
11.4.3	Approaching the $G^\#$ Problem	54
11.4.4	The $G^\#$ Problem: A Chain of Type-G Meta Problems	56
11.4.5	Optimized $G^\#$ Problem	56
11.4.6	Practical Stopping Point in the $G^\#$ Algorithm	57
11.5	Overall Optimization: $G^{\#\#}$ Problem	58
11.5.1	The Final Issue	58
11.5.2	Regression Methods for Solving the $G^{\#\#}$ Problem	59
11.5.3	Functions That Model a Given Data Set	60
11.5.4	The $G^{\#\#}$ Problem	61
11.5.5	The $G^{\#\#}$ Algorithm Employing Regression-Analysis Technology	61
11.6	Implementation Process	61
11.6.1	The Basic Implementation Alternatives	61
11.6.2	The First Implementation Alternative (Basic Implementation of the G-Algorithm)	61
11.6.3	The Second Implementation Alternative (Implementation of the $G^{\#\#}$ Algorithm)	63
11.7	Estimation Function: Operational Use	64

11.8 Which Regression Analysis Algorithm?	65
11.9 Final Remarks	65
12 Revisions and Changes Digest	65
References	67
Index	69

List of Tables

1 Space Network Forward and Return Link constraints from the Space Network Users' Guide (SNUG)	28
2 Ground Network Forward and Return Link constraints	28

List of Figures

1 Communications events placed relative to the 90-minute-orbit time line	11
2 An example prototype event for a (more) realistic communications scheduling scenario	12
3 An example illustrating the skip-factor concept	25
4 Best-Solution Fitness modeled as a function of algorithm iteration count, with its derivative	45
5 Best-Solution Fitness versus iteration count	45
6 Best-Solution Fitness as a function of parameter n , the change in the number of new members added each step	46
7 Derivative of best-solution fitness as a function of n , the change in the number of new members added each step	47
8 Best-solution fitness modeled as a function of elapsed run time	48

This page intentionally left blank.

Abstract

NASA's space data-communications infrastructure—the Space Network and the Ground Network—provide scheduled (as well as some limited types of unscheduled) data-communications services to user spacecraft. The Space Network operates several orbiting geostationary platforms (the Tracking and Data Relay Satellite System (TDRSS)), each with its own service-delivery antennas onboard. The Ground Network operates service-delivery antennas at ground stations located around the world. Together, these networks enable data transfer between user spacecraft and their mission control centers on Earth. Scheduling data-communications events for spacecraft that use the NASA communications infrastructure—the relay satellites and the ground stations—can be accomplished today with software having an operational heritage dating from the 1980s or earlier. An implementation of the scheduling methods and algorithms disclosed and formally specified herein will produce globally optimized schedules with not only optimized service delivery by the space data-communications infrastructure but also optimized satisfaction of all user requirements and prescribed constraints, including radio frequency interference (RFI) constraints. Evolutionary algorithms, a class of probabilistic strategies for searching large solution spaces, is the essential technology invoked and exploited in this disclosure. Also disclosed are secondary methods and algorithms for optimizing the execution efficiency of the schedule-generation algorithms themselves. The scheduling methods and algorithms as presented are adaptable to accommodate the complexity of scheduling the civilian and/or military data-communications infrastructure within the expected range of future users and space- or ground-based service-delivery assets. Finally, the problem itself, and the methods and algorithms, are generalized and specified formally. The generalized methods and algorithms are applicable to a very broad class of combinatorial-optimization problems that encompasses, among many others, the problem of generating optimal space-data communications schedules.

General Terms: Scheduling, Algorithm, Computing, Space

Additional Key Words and Phrases: System specification, formal mathematical specification, space-data communications, radio frequency interference mitigation, combinatorial optimization, genetic algorithm, evolutionary programming, probabilistic search, regression analysis



1. Introduction

1.1. Background on NASA Data-Communications Scheduling

Scheduling data-communications events for spacecraft that use the NASA communications infrastructure [26]—the relay satellites [19] and the ground stations [7]—can be accomplished today with software having an operational heritage dating from the 1980s or earlier [10, 31] with emphasis on incremental and reactive scheduling [22]. Neither the legacy scheduling system nor any subsequent system described in the public literature possessed the capability to generate true optimized schedules for reasons that will be explained below, but the scheduling system described in publicly available documents can (when the option was invoked) generate schedules free of radio frequency interference (RFI) effects by blocking out portions of the problem-solution space from consideration whenever those portions appeared with any predicted RFI effects. Similarly, the algorithms implemented in the legacy system pruned away portions of the solution space upon encountering violations of the various other constraints. This approach, which perforce ignores large portions of the solution space, necessarily means that true schedule optimization was not an actual achievable objective of the legacy scheduling system.

Present space data-communications schedulers commonly have the capability of algorithmically generating schedules using techniques for representing and exploring the problem-solution space as either a graph or a tree of related sub-solutions. A number of standard algorithms for searching a solution space represented as a graph or tree are available. These, in general, operate by eliminating branches of the tree where constraint violations (e.g., unacceptable levels of RFI) are found (although any “slash and burn” approach or any “branch and bound” technique has the undesirable consequence that it incorporates no mechanism by which to avoid discarding sections of the tree that represent solutions that are better than any others that can be found in the course of running the scheduler). NASA’s legacy scheduling system, using such standard search methods, was capable of producing *workable* schedules, albeit with certain significant concessions to the compute-intensive nature of the search (including certain problem simplifications that themselves, even ignoring the performance of the search techniques, precluded the possibility of true schedule optimization). See Revisions and Changes Digest item 1, page 65.

1.2. Toward an Optimizing Scheduler

Current space data-communications scheduling systems, which lack a true schedule-optimization capability,

leave open the possibility that new methods for searching the solution space might result in improved infrastructure performance and overall schedule-quality improvement, bringing increased overall customer satisfaction.

Disclosed and formally (mathematically) specified herein are methods and algorithms for an optimizing, constraint-satisfying, automated scheduling system that potentially could be implemented in NASA’s space-data-communications infrastructure. Conceived and developed in the early 1990s, this is the first such system (i.e., methods and algorithms) disclosed publicly that—

1. is capable of true schedule optimization considering prescribed constraints such as mitigation of RF interference,
2. is specified rigorously, and
3. is implementable with adequate performance.

The system (the methods and algorithms taken together) addresses (a) the issue of performance and efficiency of the communications infrastructure and (b) the important space-mission operations-planning problem of optimizing data-communications schedules. The desired optimization not only would include minimizing radio frequency interference effects that can reduce achievable data rates for space missions, but also would include accommodating other prescribed constraints such as hours of operation of mission control centers and anticipated or planned resource outages.

1.3. Incorporating RF Interference-Mitigation Constraints

Practically all of the major emitters that produce RFI effects are known as to both location (or dynamic position in space) and signal characteristics (frequency, power, signal structure, polarization, etc.). There are many such sources of RFI: ground-based radars, radio and television transmitters, other spacecraft, and even cellular telephone service-provider towers and their customers, among other emitters. (Individual cell phone users are each insignificant, but in the aggregate, they represent an RF noise “floor” that can be characterized, predicted, and mitigated in various ways.)

Beyond interference mitigation techniques (e.g., spread-spectrum signal structures that are built into both the NASA data-communications infrastructure and a user spacecraft’s onboard hardware and software), various RFI-avoidance mechanisms can be invoked during the process of scheduling communications-service delivery to users. Some of these mechanisms do not lend themselves to automation, while others are based on rules of thumb and problem

simplifications that, in general, do not promise the best possible schedules (in terms of optimizing user satisfaction and service delivery by the data-communications infrastructure), even if automated. The former category (i.e., manual mechanisms) does not represent practical approaches for NASA, but an automated mechanism in the latter category (the application of rules of thumb and problem simplifications) has been used in NASA's scheduling system at least since the operational date of the Space Network. The herein-disclosed algorithm represents a third category of techniques for RFI avoidance in schedule generation: it poses no difficulty for automation, is not based on rules of thumb (but, instead, closed-form calculations of interference effects), and supports true schedule optimization.

An optimizing scheduler that satisfies constraints on RF interference requires prior RF analysis of the factors involved in the delivery of data-communications services, including the RF environment in which the communications occur as well as the user's requirements and the specific characteristics of the user spacecraft. In the early 1990s, NASA's Communications Link Analysis and Simulation System (CLASS) [8] Communications Link Analysis and Simulation System (CLASS) fielded an interference analysis system [15]—the CLASS IAS. The CLASS software is able to produce all of the auxiliary data needed as input for the operation of an interference-mitigation scheduling system (see Section 3.1 for a representative list of these inputs).

1.4. Concerning Processes, Methods, and Algorithms

Although it represents one of the most essential concepts in computer science, the term “algorithm”, perhaps surprisingly, has no generally accepted technical (or formal) definition¹. Further, definitions of “process” and “method” typically are non technical and imprecise. Trying to identify, with accuracy, distinctions between these three terms therefore would be adventurous. However, for the purposes of this paper, we see (at least minor) conceptual differences, and these differences are reflected in the way the terms are used herein.

A process is “a series of actions or steps taken in order to achieve a particular end”².

A method, considered as a “process by which a task is completed; a way of doing something”³, entails a list of steps

to be performed to accomplish an objective or intended result. An algorithm, of course, also includes a list of steps to be performed. For the purposes of this paper, it is assumed that every algorithm, but not every method, can be performed mechanically, at least in principle. Consequently, every algorithm, but not every method, must be specifiable with precision sufficient to make it accurately implementable as a computer program.

In this paper, we endeavor to adhere to the following scheme: a sequence of steps or actions is said to be a(n)—

1. “Algorithm” when the steps are expected to be implemented in and performed by a computer application and the result of the completion of the sequence of steps is representable as a data structure.
2. “Method” when the goal of the sequence of steps is broad or high-level and a human must perform at least one of the steps.
3. “Process” when each of the steps is definite and precisely specifiable, when a human must perform at least one of the steps, and when the result of performing each of the steps in the sequence is not necessarily representable as a data structure.

1.5. The Essential Technology Used in the Present Disclosure

The algorithms and methods disclosed herein apply principles from the computer-science field of evolutionary search and combinatorial optimization [2, 4, 9, 30] to solve the problem of finding an optimal overall schedule [25, 32, 12] that (a) will satisfy user requirements for communications support from NASA's communications infrastructure and (b) will be consistent with NASA's Space Network User's Guide (SNUG) [19] as well as the operations guidelines for the Ground Network [7]. The quality of the schedules generated by a computer application program that implements this algorithm will be a monotonic function of the program's execution time. During the search for better and better solutions, the best solutions found so far are retained and given a chance to influence the creation of new, possibly better solutions. When execution is terminated arbitrarily, the output will be the best solution found so far during the run, and the longer the run, the better the solution is expected to be. More will be said regarding the notion of optimization (see Section 2.4.7 (page 15)).

In the remainder of this paper, any mention of “the disclosed algorithm” will, depending on context, be understood to mean “the disclosed methods and algorithms”.

1 Wikipedia article entitled “Algorithm” at <http://www.wikipedia.org/> (accessed 18 October 2009).

2 Oxford American Dictionaries accessed 28 October 2009.

3 Wiktionary entry at <http://en.wiktionary.org/> (accessed 28 October 2009).

1.6. Innovations Disclosed

The present disclosure incorporates well-known technologies, namely, genetic algorithms (evolutionary search) and regression analysis (in the broad sense encompassing the general concept of the mathematical modeling of relationships between data). The primary contributions disclosed herein are as follows:

1. formal specification of methods, algorithms, and processes for employing evolutionary-search technologies in an optimizing scheduling system for the NASA space-data communications infrastructure;
2. formal specification of a method and algorithm for optimizing the internal parameters of a genetic algorithm (specifically the algorithms identified in item 1); and
3. a description and formal specification of a method and algorithm for deriving a function that, given an arbitrary problem scenario in the problem domain identified in item 1, will, in a cost-effective manner, return an estimate of the optimal values of the internal parameters of a genetic algorithm for solving the given problem scenario.

The author is not aware of any other disclosures equivalent to items 1, 2, or 3.

Further, the methods and algorithms described and specified formally in Appendix B (Section 11 (page 49)) generalize those in items 1, 2, and 3 above and are applicable to a very broad class of problems. Many of the problems in this general class (referred to as problems of Type G) pertain to NASA and the space program (e.g., the space data-communications scheduling problem treated in the main body of this paper, as well as space-mission design optimization and spacecraft-design optimization). But numerous other fields (particularly those related to design optimization, and, more broadly, virtually any field where actual solutions cannot be computed directly with closed-form techniques but can be described as finite data structures that *can* be evaluated as to “goodness”) are encompassed under the generalized problem of Type G as defined in Appendix B. Finally, the formal (mathematical) specifications of the methods and algorithms are sufficiently rigorous and detailed to facilitate not only the process of relating them to this broad class of real-world problems, but also the process of implementing them in a computer-application program. (See Revisions and Changes Digest item 2, page 65.)

1.7. Audience for the Present Disclosure

As indicated above, the main body of this paper concerns a particular application (NASA space-data communications

scheduling), while Appendix B broadens the topic to encompass a very broad range of application domains. The audience for the former would include groups responsible for designing and implementing space-data communications scheduling systems (or identifying appropriate technologies that could be used in such systems), while for the latter the audience would include practitioners generally and, especially, groups designing and implementing any system for reaching optimal solutions for the generalized problem domains of Type G as defined in Appendix B. Any interest in this paper on the part of researchers and academics would likely be limited to the possible application, described herein, of well-known techniques from the field of evolutionary programming, and from the field of regression analysis generally.

1.8. Organization of Paper

Section 2 describes the scheduling-problem domain in terms of the question of tractability and identifies a viable approach to searching for optimal solutions. Such an approach (to the author’s knowledge) has not been implemented or considered for use in NASA’s space-data communications infrastructure, which is the context (or, rather, the primary context) of the present disclosure. Section 3 presents the assumptions underlying the specification of the disclosed algorithms. Section 4 defines general domain terms and specific notations used in specifying the algorithm. The algorithm itself is precisely specified in Section 5. Section 6 presents an additional algorithm (also based on the principles of evolutionary search) for optimizing the search itself—producing the optimal choice of the values of the schedule-generation algorithm’s internal parameters. In a further abstraction of the overall space data-communications scheduling problem, Section 7 outlines approaches for determining a function by which to estimate the optimal choice of the values of the schedule-generation algorithm’s internal parameters, given a scheduling scenario. The prototype implementation of the schedule-generation algorithm and the internal parameter optimization algorithm are briefly mentioned in Section 8. Concluding remarks are given in Section 9. Section 10 (Appendix A) considers a possible model to describe the performance of the schedule-generation algorithm, and describes a key insight afforded by analysis of the model. Finally, Section 11 (Appendix B) describes and specifies, in abstract terms, a broad class of problems—one member of which is the scheduling problem of the kind targeted by the methods and algorithms that are presented in the main body of this disclosure—and discloses and specifies generalized methods and algorithms for solving problems in this general class.

2. Space-Data Communications-Scheduling Problem Definition

2.1. Space-Data Communications Scheduling

2.1.1. Scheduling-System Objective

In the overall data-communications scheduling problem, the primary objective is to find a solution (i.e., a schedule) that (a) maximizes delivery of services to users according to their requirements and (b) maximizes the utilization of NASA service-delivery assets⁴.

The evaluation of candidate solutions will involve numerous factors that will be described in subsequent sections. In addition to various technical factors, the evaluation may also reflect NASA policy-level considerations such as the relative priority officially assigned to each spacecraft or mission.

2.1.2. Primary Factors Affecting Achievability of Objective

Numerous factors bear on the achievability of the primary objective. For example, the service-delivery assets in the data-communications infrastructure are subject to outages, both planned and unplanned. Planned outages result from equipment maintenance and upgrade activities. Unplanned outages are relatively rare and would include service interruptions due to severe weather or natural calamities like earthquakes. By definition, the process of searching for a solution of the overall scheduling problem does not incorporate unplanned outages.

2.1.3. Technical Factors Affecting Achievability of Objective

Technical factors, as well, bear on the achievability of the primary objective. These factors include any phenomenon or circumstance that potentially could measurably degrade data-communications performance:

1. radio frequency interference
2. signal multipath interference

3. signal-to-noise ratio reduction by noise due to spacecraft charging⁵, antenna blockage, atmospheric effects, rocket plume effects, etc.

The effects of all of these factors are dynamic, but they can be predicted through link analysis and simulation techniques [8, 15], given a user's planned spacecraft orbit and attitude profile to enable the determination of the effects listed above as items 2 and 3. Item 1 is determined by the number and characteristics (including the orbital parameters and attitude profile) of all other spacecraft (not only US spacecraft but also the spacecraft flown by other nations). The predictability of the listed effects suggests that the process of searching the solution space could be made more efficient if all candidate schedules at least avoided communications events rendered useless by the above factors. This strategy, described in general terms in [25, 33] in relation to spacecraft mutual interference, is an integral part of the algorithm disclosed herein and is applicable in general to all other predictable phenomena and circumstances that potentially could measurably reduce data-communications performance. Following this strategy, the herein-disclosed algorithm mitigates these effects automatically and produces optimal solutions of the overall scheduling problem. In the foregoing, the word "spacecraft" should be broadly interpreted to include user assets of other kinds (e.g., habitats or surface rovers).

2.2. Size of the Solution Space

2.2.1. Determining Factors

It is instructive to estimate, or at least establish a lower bound on, the size of the solution space for the data-communications scheduling problem for some simple combinations of users, user requirements, and service-delivery assets. Analysis of simple scheduling scenarios leads naturally to a better appreciation of how large might be the solution space for realistic scheduling scenarios. For any given scheduling scenario, the solution space comprises all possible schedules, each satisfying at least one requirement of at least one user.

Each schedule consists of a set of communications events. Each event represents partial satisfaction of a user requirement and is defined in terms of a number of parameters (most of which will not be discussed further herein):

- start and end times for each forward link (when a NASA support antenna is radiating signals to the user asset)

⁴ The possibility of requirements for cross communications (between user spacecraft, or between infrastructure assets) has not been overlooked and is not unimportant in the foreseeable future. Such requirements are beyond the scope of the present disclosure, but could be readily included in future adaptations of the methods and algorithms specified herein.

⁵ John Kennewell and Andrew McDonald, "Satellite Communications and Space Weather", Australian Government, Bureau of Meteorology, Ionospheric Prediction Service Radio and Space Services, Space Weather Agency web site, <http://www.ips.gov.au/Educational/1/3/2> (accessed 20 August 2008).

- start and end times for each return link (when the user asset is radiating signals to the NASA support antenna)
- the frequency selection for each forward link
- the frequency selection for each return link
- the polarization of each forward link
- the polarization of each return link
- the data rate (or symbol rate) of each forward link
- the data rate (or symbol rate) of each return link
- pseudo-random-noise (PN) spread indicator for each forward link
- pseudo-random-noise (PN) spread indicator for each return link
- NASA support-antenna selection for each forward link
- NASA support-antenna selection for each return link

2.2.2. A Simple Example

The start and end times for a scheduled event have a one-second granularity and are represented as seconds of offset from some prescribed epoch (usually specified as a reference date such as 1 January 1970 that is “over the horizon” of past time in the current context). For a normal two-week scheduling period, there are $14 \times 24 \times 60 \times 60 = 1,209,600$ possible values for a start or end time. The event-duration minimum is nominally ten seconds and the duration maximum is nominally 80 minutes. For a given start time, there would be $80 \times 60 - 10 = 4790$ allowed intervals each representing a communications event at the given start time without violating the 80-minute maximum limit or the 10-second minimum limit. If a schedule had only a single communications event with a single link for a given user during the two-week scheduling period, with no other constraints, there would be approximately $4790 \times (1209600 - 4790/2)$ or 5.78×10^9 allowed instantiations of the event without violating the 80-minute maximum limit or the 10-second minimum limit. In a two-week scheduling period, with nominal duration of 90 minutes per orbit, there would be $(14 \times 24 \times 60)/90 = 224$ orbits. If in a two-week scheduling period, the given user required one single-link event in each orbit (see Figure 1), there would be 224 events in the schedule. In each orbit, there would be approximately $60 \times (4790 \times 10 + 4790 \times 80/2) = 14,370,000$ possible events without violating the 80-minute maximum limit or the 10-second minimum limit. The number of possible schedules—that is, the number of possible combinations of the possible events (with exactly one event in each of the 224 orbits) (ignoring other constraints)—would be approximately $(1.437 \times 10^7)^{224}$. Thus, for this trivial scheduling scenario, the scheduling problem would have more than

10^{1603} possible solutions. Even if the maximum allowed duration of each event were reduced to 10 minutes, the cardinality of the solution space would still exceed 10^{1450} . For perspective, consider this number in relation to the number of neutron-size spheres that could be packed into a sphere the size of the known universe—a number on the order of 10^{135} .

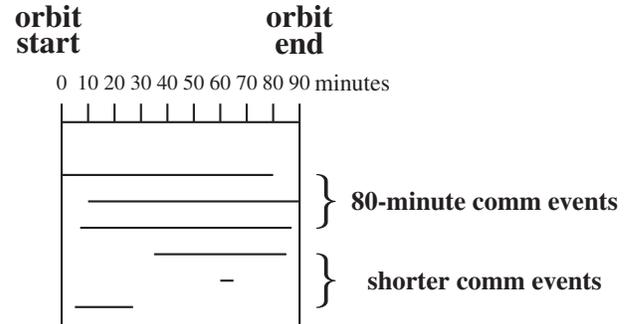


Figure 1. Communications events placed relative to the 90-minute-orbit time line, as allowed in the simple example scheduling scenario. Three possible maximum-duration communications events are shown, each with a different start time offset from the start of the orbit. Three other possible communications events are shown with different allowed durations and start-time offsets. In the simple example scenario, the user requires only one communications event in the orbit.

2.2.3. A More Realistic Example

The next factor bearing on the estimate of the size of the solution space for the TDRSS scheduling problem is the concept of the prototype event. A prototype event (to be defined more exactly in Subsection 4.3 (Definition 53 on page 23)) is, in general terms, a combination of data-communications link activations for actual forward and return data flows by which a user will accomplish various purposes, including forward links for delivering commands, data, and software loads to the spacecraft. Other purposes will pertain to returning data via return links to the mission control center or to scientists. Each of the links is defined in terms of parameters that specify data rate, frequency, polarization, support antenna, etc. The specification of the values of these parameters, along

with the start and stop times in seconds of relative offset, and the allowed tolerance in these offsets, makes up the definition of a prototype event.

If, in a given scheduling-problem scenario, a prototype event has a required duration of 45 minutes and consists of five links, each with five allowed choices of NASA service-delivery antennas, and each with nominal duration of 10 minutes, start-time tolerance of three minutes, and duration tolerance of three minutes, then in comparison with the above example (a single spacecraft requiring one event per orbit), the size of the solution space (or its lower bound) would become a much larger number. This number is roughly calculated by first calculating the number of possible instances of each prototype event that could be instantiated for any given prototype-event start time, and then calculating the number of possible instantiations of the prototype event in a given orbit. Allowing any start time that does not exceed $45 - (10 + 3) = 32$ minutes of offset from the prototype-event start time, the number of possible instantiations of one of the links (ignoring the choices for service delivery antenna) is $60 \times 32 \times 60 \times ((10 + 3) - (10 - 3)) = 691200$. Allowing any start time offset between 32 minutes and $45 - (10 - 3) = 38$ minutes from the prototype-event start time, the number of possible instantiations of one of the links (ignoring the choices for service-delivery antenna) is approximately $60 \times (38 - 32) \times 60 \times ((10 + 3) - (10 - 3))/2 = 64800$. The minimum allowed duration of a link represents a constraint to ensure that a link is not allowed to start with an offset of more than (in the present example) 38 minutes. Thus, for a given instantiation of the prototype event, the total number of allowed instantiations of one of the links (ignoring the choices for service-delivery antenna) would be $691200 + 64800 = 756000$.

To simplify calculations and yet establish a lower bound on the number of possible instantiations of the given prototype event, assume there is only one allowed start time for a prototype event in each orbit (although, typically, the allowed start time for a 45-minute prototype event would be any time in the first 45 minutes of the 90-minute orbit, so that there would be 2700 different allowed start times in each orbit). Then the number of possible instantiations of the prototype event per orbit—each instantiation being an allowed combination of five prototype-event links—would be $756000^5 = 2.469 \times 10^{29}$. Therefore, the number of possible combinations of prototype events in the 224 orbits in the scheduling period would exceed $(10^{29})^{224} = 10^{6496}$.

Thus, with consideration of all possible solutions allowing all possible combinations of prototype events for multiple users, with all possible combinations of allowed choices of data rate, frequency, polarization, support antennas, etc., it quickly becomes apparent that the solution space for a realistic scheduling scenario would be much larger yet.

Prototype event

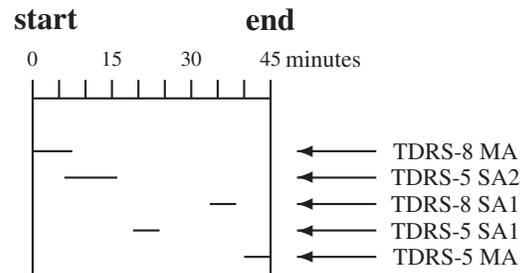


Figure 2. An example prototype event for a more realistic communications scheduling scenario. The user's specification for a prototype event requires five links established and completed between the start time and end time of the prototype event. Each link has allowed tolerances on start time and duration, and has one of five allowed service-delivery antennas assigned to it (e.g., the TDRS-5 SA2 antenna). The user requires that an instance of the prototype event is to be scheduled relative to some prescribed mission event (e.g., the start of an orbit), with some prescribed tolerance on the start time.

2.2.4. Possible Approach to Reducing the Size of the Solution Space

A reduction of the size of the solution space could be realized by increasing the granularity of the allowed start times and durations for events. For example, one might allow only even-numbered seconds of offset from the prescribed epoch. At best, this would reduce the size of the solution space in the above simple example by a factor of $(2^{-5})^{224} = 2^{-1120}$, or 10^{-337} , from 10^{6496} to a number that yet still exceeds 10^{6159} . Such an approach, if it results in a significant reduction in the size of the solution space, will not effect a tractable problem without at the same time effectively eliminating the possibility of finding optimal solutions. Making optimal solutions less likely (or impossible) to be found does not represent an attractive characteristic of an approach for reducing the size of the solution space: enabling worse solutions to be found faster presents a dubious gain.

2.3. Alternative Approaches

2.3.1. Brute Force and Constructive Techniques

From the very large size of the solution space—even for very simple scheduling scenarios as trivial as the above examples—it becomes clear that the feasibility of finding optimal schedules will depend on the feasibility of a method that does not rely on brute-force search (i.e., a search strategy that requires the examination/evaluation of every possible solution (i.e., every possible schedule)). Inescapably, only an exceedingly large number could represent the size of the solution space for real-world scheduling scenarios for the actual users of NASA’s space data-communications services, and the idea of examining all of the possible solutions is untenable without resorting to some truly exotic method, such as the yet-to-become-practical idea of quantum computing. The general problem of creating *optimal* schedules to satisfy users’ data-communications requirements cannot be solved using a single, general prescriptive formula, nor with brute-force search through the solution space, even with powerful computers (possibly excepting future quantum computers) and the most sophisticated graph or tree-traversal algorithms (such as the A* algorithm [3, 22] (not to be confused with the set of all antenna IDs (Definition 36 (page 21))))). From the foregoing examples, it will have been seen that the general optimal-solution search problem cannot be attacked with such approaches.

However, *workable* solutions can be generated with the techniques that are in use in the current operational scheduling system, and these solutions are free of violations of constraints (e.g., RF interference, if this option is activated in the scheduler). However, there is no expectation that the current operational approach will produce solutions that are near optimum. Further, since an optimum solution (in the absolute sense) would necessarily remain unknown, it cannot be determined how nearly optimum these found solutions might be (and even if an upper bound on the difference could be calculated, it would be without benefit, since the approach of the present operational system offers no way to use the information to produce better schedules).

2.3.2. Necessity of Special Search Techniques

From the foregoing, it becomes a convincing proposition that neither a brute-force strategy nor the approach of the current or any previous NASA scheduling system can offer the possibility of discovering an optimal solution for realistic scheduling scenarios: other techniques are required. While brute-force search performed using quantum-computing techniques might be explored in the future, a more immediately available approach is worth considering for the interim. Research and

application experience since the 1980s has resulted in establishing the viability of probabilistic search strategies for certain types of optimization problems that have very large solution spaces. In particular, genetic algorithms [2, 4, 9, 30] (or, more generally, evolutionary algorithms) have been successfully applied to scheduling and planning problems [1, 5, 13, 17, 28, 29, 32]. While other probabilistic search strategies (including those under the heading of regression-analysis techniques) are invoked at a high level herein (see Sections 7.2 and 11.5.2), their detailed treatment is beyond the scope of this disclosure.

2.4. Evolutionary Search

2.4.1. Genetic Algorithms

Genetic algorithms belong to a well-studied class of algorithms abstractly inspired by biological selection and generation, individual inheritance and mutation, species adaptation, species survival, and fitness [11]. Both in the algorithmic realm and in biology, selected individuals in each generation mate and produce offspring. The offspring have some but not all of the traits of their parents, and in fact have new traits as a result of genetic crossover and mutation. Individuals die and make way for individuals in the new generation, who will survive better or not, depending on their fitness for survival in their given environment—which environment will favor the more-fit individuals with more power to produce offspring for the next generation. Over successive generations, the principle of “survival of the fittest” implies the expectation that the survival of individuals will improve overall, i.e., the average fitness of members of the population will improve progressively.

2.4.2. Representations of Solutions of the Scheduling Problem

The present disclosure invokes the principles of genetic algorithms to attack the problem of scheduling space-data communications, with the incorporation of the further, secondary objective of minimizing the effects of RF interference in the optimization. (Note that this further objective represents a primary example of the general constraint-satisfying capabilities of the disclosed algorithms and methods.) Under the conceptual description given above, a solution, by definition, is a schedule. The question of how to represent a solution for the purpose of searching the entire solution space for an optimal solution is fundamentally important, and has an immediate answer. The answer, since every solution in the solution space is a schedule, is that a solution is a finite data structure in computer memory that represents a schedule, which, by definition, is simply a list of communications events each

of which at least partially satisfies at least one user requirement for communications services. The combined data structure that represents the whole evolving population of candidate solutions at a given iteration of the running search process is just the combined list of individual members of the current population. The data structure for any individual (being one possible solution of the entire scheduling problem) is subjected to evaluation by a fitness function that determines the individual's survivability or fitness, i.e., the degree to which the individual measures up to prescribed criteria for selection as a member of, or to produce offspring for, the next generation.

2.4.3. Processes for Selection and Creation of Working Population Members

The fitness function is the essence of the problem to be solved by the genetic algorithm and remains unaltered for the duration of the entire search for a solution. Each member (i.e., each schedule) of a given generation would have a fitness score determined by the prescribed fitness function. In each generational cycle, to prepare a new generation, a prescribed process (sub-algorithm) operates to perform a selection of individuals (schedules) to survive into the new generation and/or to be used to generate offspring. Another prescribed process (sub-algorithm) then (a) creates the offspring of the selected individuals by mathematically transforming and combining their traits (i.e., the values in the data structure representing an individual), and (b) adds new members using an algorithm for randomly generating new schedules, to effect the probabilistic exploration of the entire solution space. The probabilistic nature of the exploration arises from the use—during the execution of both the selection process and the new-member-creation process—of random-number generators in the computing platform on which the genetic algorithm executes. More will be said in Section 2.4.9 concerning random numbers in relation to the algorithms disclosed herein.

2.4.4. Principle of Operation of Genetic Algorithms

Operation of the genetic algorithm begins with the creation of an initial population of some size. The manner in which the members of the initial population are generated can affect the search progress, although the nature of the algorithm tends to diminish the effect over the course of the search. A new generation results from applying a variety of mathematical transformations not only to individuals (i.e., their data structures), but also to pairs of individuals in the current generation, which each will already have been evaluated by the fitness function during the process of selecting the members of the previous generation to compose the current generation.

The more-fit individuals, being relatively favored (and, consequently, more likely themselves to persist for multiple generations), will have relatively more offspring than will result from less-fit individuals. However, randomly selected individuals will also produce offspring at some rate, and new individuals randomly created are also introduced into the population at some rate. The new generation will then undergo the same process of evaluation and transformation to result again in a new generation. After some number of repetitions of the foregoing generational process, the current generation will, with some likelihood, include individuals that are more fit than the best members of any previous generation. Eventually, after many generations, the expectation is that an additional iteration of the evolutionary search process will have a vanishingly small likelihood of producing further significant improvement of the best individuals of the new generation over those of the previous generations (see Footnote 6 (page 46)).

2.4.5. Progress of Evolutionary Search

After an expected initial rapid improvement in the fitness of the best individuals, each successive generation will see, on average, less and less improvement. There occasionally can be sharp improvements from one generation to the next. Such improvements can occur by virtue of the probabilistic nature of the genetic algorithm-based search strategy: occasionally, during the search, a newly created individual (schedule) will, by chance, be much better (as evaluated by the fitness function) than any other individual found earlier in the search. In this way, the algorithm can find and explore another region with a local minimum/maximum—which, with some probability, *could* be the global minimum/maximum. However, the search effort/time required to produce significant *new* improvement in the best individuals eventually will become untenable. Ultimately, the search must be terminated and the best individual (or rather one of possibly multiple individuals having the same best value of the fitness score) would then be used as the search result—with no guarantee, however, that this solution is in fact the *absolute* best, and, furthermore, with no good way to show that it is *not*.

Exploration of the solution space using a genetic algorithm may entail certain difficulties that have been described in the literature, e.g., “premature convergence” and “genetic drift”. A determination of the degree to which such difficulties might be present in the disclosed methods and algorithms has not been undertaken and is beyond the scope of this paper. However, the disclosed methods and algorithms include the S^d Algorithm (Section 6 (page 37)) that directly addresses these issues, effectively “tuning” the genetic algorithm that finds an optimal solution for a given space-data-communications-scheduling problem scenario. Further, Sub-

section 11.4 (page 54) presents methods and algorithms that effectively address similar issues relative to solving the generalized problem.

2.4.6. Effect of Internal Parameters of the Genetic Algorithm

A genetic algorithm, in the general case, has internal parameters that affect the efficiency and effectiveness of the search, including—

- the size of the initial population
- the size of subsequent generations
- the proportion of each generation that is selected randomly for survival into the next generation
- the number of offspring produced by selected individuals
- the number of randomly created individuals added to each new generation
- the mutation rate, (i.e., the number of new members created in each new generation via the mutation mechanism)
- the crossover rate, (i.e., the number of new members created in each new generation via the crossover mechanism)
- the number of “gene” crossover points (where, informally, “gene” refers to values in the data structure that specifies an individual member of the population)

It is natural to ask how best to set the values of the internal parameters when the objective is to run an implementation of the algorithm and achieve the greatest possible degree of effectiveness and efficiency in generating an optimal solution to the scheduling problem. Some understanding of the effect of different combinations of the possible values of the parameters can be gained through systematic experimentation—testing different combinations to reveal how they affect the efficiency and effectiveness of the search process. Such a process, if performed manually, would be time consuming and tedious. It also might present conceptual and theoretical difficulties relative to drawing reliable conclusions.

By asking what might be the best combination of the values of the internal parameters, one then sees another, somewhat more abstract optimization problem. The author’s implementation of the unpublished predecessor of the herein disclosed algorithm included the use of a genetic-algorithm approach to solve this problem as well, to derive an optimal combination of those values. Details of this optimization approach are found in Section 6 for the schedule-generation algorithm to be presented in Section 5 and (relative to the generalization of the algorithms and methods that will be pre-

sented in the main body of this disclosure) in Appendix B (Subsection 11.4 (page 54)).

2.4.7. Fitness Functions, Minima, Maxima, and Optima

The fitness function applied to an individual (i.e., a schedule) in a given generation will produce a numerical value. In principle, the fitness score could be determined for every schedule in the entire solution space. If this were done, a kind of hypersurface (representing the fitness score (the dependent variable) as a function of the schedule (the independent variable)) could be constructed and analyzed mathematically, and could be visualized as having contours, peaks, and valleys. The visualization of the function would also exhibit discontinuities resulting from, for instance, the discreteness of the allowed values for many of the parameters in the definition of a communications event. The sheer magnitude of the size of the solution space makes such a surface infeasible to construct, but the very class of probabilistic exploration algorithms we are concerned with in this disclosure could (if desired) even be used to approach the question of characterizing how “nice” the hypersurface is. In any case, the disclosed algorithm is well suited to exploration of the whole solution space, “nice” or not.

The essence of the optimization question pertains to finding the global maximum (or the global minimum, depending on how the fitness function is defined) of the search space. No known practical method exists to reach an “absolute” answer to this question for the general case. At this writing, no other available method has been shown to surpass the results that can be attained through a method based upon evolutionary (probabilistic) search: no known non-probabilistic method for addressing the general scheduling problem has been shown to be capable of surpassing the results that can be attained using available probabilistic search methods.

The term *optimizing*, as applied to the algorithms described herein, reflects this aspect of directed, iterative probabilistic search, where the search space is probed by a random process to find more and more minima/maxima (local as well as global), and by another process that gives each of the best candidate solutions from the current generation a chance to have “children” that are still *more* fit as solutions to the scheduling problem.

In general, the nature of the problem, with its extremely large solution space, leaves open the possibility that, at the termination of any arbitrarily long run of the application program on a computer, a “better” solution *might* have been found by letting the application produce just one more generation. When the run is terminated, the best solution found is considered to be optimized in terms of the probabilistic coverage of the entire search space. This is the primary sense in which the term “optimal” is used in this disclosure, but there

is, in a practical vein, an additional consideration, namely, the cost of the process of searching through the solution space for the best possible solution. Absent any limit on the speed or storage capacity of computing resources, there would be no trade-off between solution quality and search time. *Actual* limits on computational speed and storage will necessarily mean that, for any computer application that solves the kind of optimization problems addressed herein, solution optimality will be directly related to the duration of the run of the application, which can never be unbounded. Thus, the designation of “optimal” implies that enough computing power has been applied to reach a point where the law of diminishing returns (or at least a similar principle; see Footnote 6 (page 46)) would mean that a relatively large additional search effort would not have any significant expectation of improvement in the best solution found to that point. Conversely, if such a point has not been reached, then nothing can be asserted as to the optimality of any solution found.

An unavoidable issue arises from the above discussion—the question whether it could be determined in advance what computing resources would be sufficient to reach the point described in the preceding paragraph (i.e., the point where it would be legitimate to declare that no reasonable additional search time would provide any reasonable expectation of significant improvement in the solution found). No known method can answer this question, except the *empirical* method, i.e., experimentation, although the kind of algorithm-performance modeling described in Appendix A may afford additional insights.

2.4.8. Metrics for Evaluation of the Scheduling System

Possible metrics to evaluate a scheduling system include (a) determining the percentage utilization of service-delivery assets (e.g., TDRSS antennas) and (b) assessing the degree of satisfaction of customer requirements. In general, closed-form algorithms and techniques (including constructive techniques and graph-search techniques) do not afford in a scheduling system a strategy for directly optimizing, in the true sense, these metrics or indeed any other metrics. The herein disclosed algorithm, based upon probabilistic search, lends itself to incorporation of, and optimizing, a wide range of possible combinations of metrics to meet future goals for overall infrastructure performance. However, a full discussion of the possible metrics for evaluating the scheduling system is beyond the scope of this disclosure. (See Revisions and Changes Digest item 3, page 66.)

Directly comparing the current operational system with one following the present disclosure might be accomplished by evaluating schedule quality using the herein disclosed fitness function (see Definition 102 (page 32)) or a combination

of its constituent submetrics (e.g., the function **satisfied*** (see Definition 100 (page 31)), which measures the degree of satisfaction of data-return requirements of all users.)

2.4.9. Random Numbers and Their Role

Random numbers play a crucial role in the probabilistic search strategy and algorithms embodied in the present disclosure, including the schedule-generation algorithm (Algorithm 1 (page 35)) and the $S^\#$ algorithm (Algorithm 2 (page 35)), as well as algorithms in Appendix B (Section 11 (page 49)).

It is noted that “random” numbers are generated by special software or hardware on the platform that constitutes the computing resources on which an implementation of the herein disclosed algorithms will run. On typical platforms, the random-number generator implements a special algorithm that can be configured with a “seed” as the starting point for calculations to produce a random number. Repeatability of results can be assured by selecting the same seed for repeated runs—which is a feature that supports application software testing and debugging. The generated random numbers are not truly random, which is a well recognized aspect of all known algorithms for generating random numbers. Interestingly, characterizing the difference between the output of random-number generators and the output of true random processes is not a settled matter, necessarily involving arcane argumentation.

The fact that the generally available random-number generators do not produce “true” random numbers does not invalidate their use in ordinary applications (such as the one described in this disclosure), where their use is generally accepted. We accept the proposition that the use of a true (or at least the best possible) random-number generator would not improve the results or performance of a system that implemented the algorithms disclosed herein.

3. Communications-Scheduling Assumptions

3.1. Necessary Input Data

The disclosed method and algorithm assume the availability of inputs from a computational resource that provides the following information:

- predicted user-communications view periods relative to each NASA support antenna
- user-spacecraft orbit start and end times, with orbit numbers

- start and end times for intervals during which the user’s Project Operations Control Center (POCC) (or, synonymously, Mission Operations Control Center (MOCC)) is in operation
- start and end times for other relevant mission events (user-spacecraft sunrise, user-spacecraft-over-land, etc.) whenever there are active user requirements that specify any relationship to such events
- potential-interference intervals (predicted intervals during which RF signal interference would prevent NASA from satisfying a particular user requirement or request for communications services)
- intervals of predicted user-antenna blockage and multipath interference with respect to each support antenna, based on planned user-spacecraft attitude profiles
- outage intervals (predicted/planned intervals during which service-delivery resources will be unavailable)

Such a computational resource, the Communications Link Analysis and Simulation System (CLASS) [8, 15, 20], has been in operation at the NASA Goddard Space Flight Center since the early 1980s. The CLASS system was used in generating input data for runs of the prototype implementation of the predecessor of the disclosed algorithm, which implementation is described in Section 8.1.

3.2. Scope Limitations

3.2.1. Two-Week Scheduling on a Weekly Cycle

NASA’s present operational scheduling system performs the scheduling function on a weekly basis for a two-week scheduling period. The second week of the previously generated two-week schedule becomes the first week of the new two-week schedule and is adjusted by the scheduling system to reflect updated or revised information concerning user requirements, planned outages, and other factors. The second week is scheduled afresh. Although it was not a design goal of the algorithm disclosed herein, it could be revised to provide this rescheduling functionality. However, it may be appropriate to reconsider the need for a two-week scheduling cycle when a new optimal schedule could efficiently be generated weekly (or on demand) by means of the method and algorithm disclosed herein.

3.2.2. Dynamic Rescheduling

Although NASA’s present operational scheduling system can perform rescheduling under dynamic operational conditions (where, for example, a spacecraft has a declared emergency,

or a service-delivery resource has an unplanned outage), rescheduling is not considered herein and does not correspond to a design goal of the disclosed algorithms. Further discussion of dynamic rescheduling is beyond the scope of the present disclosure, but it has not been seen to present technical difficulties in a possible revised version of the disclosed methods and algorithms.

3.2.3. Near-Earth Communications Environment

NASA has been pursuing goals for crewed (as well as new un-crewed) missions that may be developed to explore the Moon and Mars over the next several decades. Such missions crucially depend on adequate communications involving RF links between the Earth and numerous remote assets including the mission vehicles and habitats. The future evolved infrastructure to provide the needed communications capabilities is in the early stages of definition, but is likely to have considerable similarities to the current space data-communications infrastructure serving near-Earth missions. For example, it is likely to have capabilities for “demand access” as well as a large reliance on scheduled communications events, where infrastructure support antennas and associated equipment would be scheduled to be configured to support user assets. While it was not a specific design goal to include the non near-Earth infrastructure in the disclosed methods and algorithms, the essential concepts already embodied in the present near-Earth infrastructure would continue to be applicable. The one major issue that would need to be addressed for the non near-Earth infrastructure pertains to RF signal latency due to the large distances involved, especially between the Earth and Mars.

4. Definitions

4.1. Basic Space-Data-Communications Definitions

CLASS— Communications Link Analysis and Simulation System. CLASS is a software system developed, maintained, and operated at NASA Goddard Space Flight Center for the purpose of supporting all aspects of space communications including spacecraft and communications infrastructure design and operations. (See Subsection 1.3 (page 7) and Subsection 3.1 (page 16).)

Communications View Period— A time interval during which a given NASA service delivery antenna is capable of being pointed toward a given user asset (spacecraft, rover, etc.), with a clear RF path that will permit data transfer using radio signals that have

prescribed characteristics (frequency, power, polarization, etc.). (See formal definition of M (and the explanation), Definition 46 in Subsection 4.3, page 22.)

Epoch— A date and time specified precisely and used as a reference time to specify later points in time as offsets from the epoch. For example, a NASA mission may specify time as seconds of offset from the epoch date and time of 00:00 Hours on 1 January 1970.

Forward— The direction of data flow from a NASA support antenna to a user asset (spacecraft, rover, etc.). (Note that this definition may admit some ambiguity under various operational circumstances using particular protocols (e.g. “acknowledgment” protocols as used in standard communications networks). The term is essentially meaningless in the context of two-way voice communications over a space communications link.)

Link— An established RF connection between a transmitter and receiver configured with compatible signal frequencies, polarization, framing, coding, and data formats, with sufficient received signal power to enable data transfer. The description of such a connection.

MA— Multiple Access; identifies the electrically “steerable”, phased-array antenna on a TDRS spacecraft. MAF/MAR (MA Forward/Return). See definition of SA.

MOCC— Mission Operations Control Center. A facility housing personnel, equipment, software systems, and other resources, with necessary communications interfaces with external entities, for the control and operation of a space mission. Synonymous with Project Operations Control Center (POCC).

Outage Interval— A planned or anticipated interval during which a service-delivery resource will not be in service. This includes intervals designated for equipment maintenance, upgrade, or calibration. (See formal definition of O , Definition 39 in Subsection 4.3, page 21.)

POCC— Project Operations Control Center. Synonymous with Mission Operations Control Center (MOCC).

Potential Interference Interval— A predicted interval during which unacceptable RF interference would affect signals received by either a user antenna or a service-delivery antenna. (See formal definition of I , Definition 49 in Subsection 4.3, page 22.)

Priority— A NASA-assigned numerical value that establishes the order of precedence of a given user relative to others, for the purpose of determining which of any two users will have precedence whenever they are in dynamic contention for NASA communications services.

(See formal definition of Φ , Definition 50 in Subsection 4.3, page 22.) The priority value is increased when a user has a declared contingency (e.g., the unexpected failure of a gyroscope on a spacecraft), although such a circumstance is not relevant for a scheduler, since by definition a declared contingency is not planned.

Return— The direction of data flow from a user asset (spacecraft, rover, etc.) to a NASA support antenna. (See remark under “Forward” regarding ambiguity of the term in certain circumstances.)

SA— Single Access; refers to the two steerable dish antennas on a TDRS spacecraft. KSAF/KSAR (K-band SA Forward/Return). SSAF/SSAR (S-band SA Forward/Return). See definition of MA.

Schedule— A collection of communications support events placed on a time line for a given time period (typically two weeks) and identified by a set of parameter values that enable the NASA communications infrastructure to be properly configured to provide communications services to users. (See formal definition of Θ in Subsection 4.3, page 26.)

User— A spacecraft or (depending on context) its associated mission project that is authorized and properly configured to make use of NASA space data-communications services. A user can also be a rover on the surface of the Moon, or a special device on the Earth’s surface designed to enable calibration of TDRSS ranging capabilities. An example of a user is the Hubble Space Telescope.

User Requirement (or User Request)— A specification of data-communications services needed by a user. Such a specification can be either specific or generic. Specific requirements give start time and end time either as absolute times (e.g., as a date and time in the Julian calendar or as seconds of offset from a prescribed epoch) or as seconds of offset from a prescribed mission event (e.g., spacecraft sunrise in orbit number 694). A generic requirement represents a repeating support service, with start and end times always defined in terms of a repeating mission event such as spacecraft sunrise. For both specific and generic requirements, the specification refers to some user-defined prototype communications event (see formal definition of C , Definition 53 in Subsection 4.3, page 23), which defines the communications links required for each instance of the event, along with other relevant parameters.

4.2. General Notation

The formal specifications of the herein disclosed algorithms depends on precise mathematical notation (in particular, *set-*

builder notation) involving a number of general terms and symbols defined in this subsection.

In set-builder notation, a set may be specified by using “curly braces” to enclose a list of its elements. For example, the set A consisting of the squares of the first five counting numbers could be specified as $A = \{1, 4, 9, 16, 25\}$. Alternatively, a set may be specified using the “set builder” notation, by which the same set could be specified as $\{x : x \text{ is the square of one of the first five counting numbers}\}$ or $\{x : \exists i \in \mathbb{N}^+ \ni i < 6, x = i^2\}$, which translates to English as “the set to which x belongs if and only if there exists i belonging to the set of all positive integers such that i is less than 6 and x is the square of i ”.

A fundamental concept in logic is that of logical negation, denoted by the symbol \neg (pronounced “not” or “being not true that”). \neg has the meaning of “not” in a logical expression. The expression $\neg x$ has a value opposite the logical value of x : that is, the expression is “true” if x is “false” and “false” if x is “true”.

(See item 4 (page 66) in Revisions and Changes Digest.)

Definition 1: \bullet (pronounced “bullet”) is a placeholder symbol representing any allowed value in the indicated place in a formula or expression, without regard to which allowed value might be chosen.

Definition 2 (Universe): Ω is the universe of discourse, i.e., the set of all objects that can be a member of a set defined in the present disclosure.

Definition 3 (Empty Set): \emptyset denotes the empty set, i.e., the set that has no member.

Definition 4 (Set of All Integers): \mathbb{Z} is the set of all integers.

Definition 5 (Set of All Nonnegative Integers): \mathbb{N} is the set of all nonnegative integers.

Definition 6 (Set of All Positive Integers): \mathbb{N}^+ is the set of all positive integers.

Definition 7 (Set of all real numbers): \mathbb{R} is the set of all real numbers.

(See Revisions and Changes Digest item 6, page 66.)

Definition 8 (Cardinality): $\forall Q \subseteq \Omega, |Q|$ denotes the *cardinality* of Q (i.e., the number of members of Q). $n = |Q| \Leftrightarrow n \in \mathbb{N}$ and Q has exactly n members.

Note that $|\emptyset| = 0$.

Definition 9 (Power Set): $\forall Q \subseteq \Omega, \wp(Q)$ denotes the *power set* of Q , i.e., $y \in \wp(Q) \Leftrightarrow y \subseteq Q$.

The power set of Q is the set of all subsets of Q . (See Revisions and Changes Digest item 7, page 66.)

Definition 10 (Set of All Finite Sets):

$$\Omega_{\mathbf{F}} \subseteq \wp(\Omega) \ni Q \in \Omega_{\mathbf{F}} \Leftrightarrow \exists n \in \mathbb{N} \ni |Q| = n.$$

$\Omega_{\mathbf{F}}$ is the set of all finite sets.

Definition 11 (Cartesian Product): $\forall A, B \subseteq \Omega$, the *Cartesian product* $A \times B = \{(a, b) : a \in A, b \in B\}$, i.e., $A \times B$ is the set of all ordered pairs $(a \in A, b \in B)$. $\forall Q \subseteq \Omega, Q^2 = Q \times Q$. $[Q \subseteq \Omega, 2 < n \in \mathbb{N}] \Rightarrow Q^n = Q \times Q^{n-1}$.

Definition 12 (Function): $\forall A, B \subseteq \Omega$, f is said to be a *function from A to B* , denoted by $f : A \rightarrow B$, if and only if $f \subseteq A \times B \ni (a, b), (a, c) \in f \Rightarrow b = c$. A *function* is considered to be a mapping from a *domain* to a *codomain*. If f is a function, **dom**(f) denotes $\{a : (a, \bullet) \in f\}$, the domain of f , and **codomain**(f) denotes $\{b : (\bullet, b) \in f\}$, the codomain of f .

Definition 13 (Set of All Functions From Y to X):

$\forall X, Y \subseteq \Omega, X^Y = \{f : [f : Y \rightarrow X]\}$, the set of all functions from Y to X .

Note that in some contexts (when not each of X and Y is a set), the meaning of X^Y differs (e.g., see Definition 56 (page 24)). Note also that X and Y in this context are free variable names, not to be confused with any domain-specific objects defined later.

Definition 14 (Function composition):

$$[X, Y, Z \subseteq \Omega, f \in X^Y, g \in Z^X] \Rightarrow g \circ f \in Z^Y.$$

The symbol \circ (read as “circle” or “composed with”) denotes “function composition”, representing the process of using the output of one function as the input to another. In performing the process for a given value a in Y (the domain of f), the function f is used to obtain the value $f(a)$ (a value in X (the codomain of f)), and this output from f is used as the input to the function g (whose domain is X) to obtain the value $g(f(a))$, which is a member of Z , the codomain of g . Thus, $g \circ f$ is a function mapping Y to Z . (Note that X, Y , and Z in this context are free variable names, not to be confused with any domain-specific objects defined later. Also, note that free variable Z is not to be confused with \mathbb{Z} , the set of all integers (see Definition 4, page 19).) See Revisions and Changes Digest item 5, page 66.

Definition 15 (set difference): $\forall Q \subseteq \Omega, \forall A \subseteq Q, Q \setminus A = \{x \in Q : \neg x \in A\}$, the set of members of Q that do not belong to A .

$Q \setminus A$ denotes *set difference*.

Definition 16 (Set of All Integer Intervals): Given integers $a, b \in \mathbb{Z} \ni a < b$, the *integer interval* $[a, b]$ is the set of all integers between and including a and b . $\bar{\mathbb{Z}}$ is the set of all inte-

ger intervals, that is, $\bar{\mathbb{Z}} = \{x \in \mathbb{Z} : \exists a, b \in \mathbb{Z} \ni a < b \text{ and } i \in x \Leftrightarrow i \in \mathbb{Z}, a \leq i \leq b\}$.

Note that in this disclosure, the notation $[a, b]$ can apply to either integer intervals or closed intervals of real numbers, depending upon context. In some contexts, square brackets are only typographical grouping marks, similar to parentheses and curly braces. See Revisions and Changes Digest item 8, page 66.

Definition 17: rndint: $\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z} \ni i, j \in \mathbb{Z}, i \leq j \Rightarrow$
rndint (i, j) is a random integer in the closed interval $[i, j]$.

rndint is a “pseudo-function” in the sense that, in any two invocations for the same arguments, it does not necessarily return the same result, assured by the use of a randomizing mechanism in the processing system on which the application is running.

A further note concerning functions is in order: except when a “pseudo-function” like **rndint** is involved, the members of the mapping (the ordered pairs) are fixed.

Definition 18 (Left-Most and Right-Most Points of an Interval): $\forall \eta = [a, b] \in \bar{\mathbb{Z}}, \eta^- = a \text{ and } \eta^+ = b$.

Definition 19 (Ordering Relation for Intervals):

$$\forall \eta, \beta \in \bar{\mathbb{Z}}, \eta < \beta \Leftrightarrow \eta^+ \leq \beta^-.$$

This is the ordering relation for intervals.

Definition 20 (Ordering Relation for Sets of Intervals):

$$\forall A, B \in \wp(\bar{\mathbb{Z}}), A < B \Leftrightarrow [\eta \in A, \beta \in B \Rightarrow \eta < \beta].$$

This is the ordering relation for sets of intervals.

Definition 21 (Sequence): s is said to be a *sequence* if and only if

$$s \in \Omega^{\mathbb{N}} \ni$$

1. $\exists a \in \Omega \ni (0, a) \in s$ and
2. $(j \in \mathbb{N}^+, \bullet) \in s \Rightarrow \exists b \in \Omega \ni (j-1, b) \in s$.

Note that the first element of a sequence has index value 0, and that no index value is skipped. See Revisions and Changes Digest item 9 on page 66.

Definition 22: For each sequence s , if $(i, a) \in s$, then a is denoted by $s_i, s[i]$, or $s(i)$.

Definition 23 (Length of a sequence): For each finite sequence s ,

1. the number of elements of s is denoted by **len** (s) and
2. s is represented as $\langle s_0, s_1, \dots, s_{n-1} \rangle$, where $n = \mathbf{len}(s)$.

Definition 24 (Tuple): $\forall n \in \mathbb{N}^+, q$ is said to be an n -tuple if and only if q is a list of objects indexed by their position in the list, where the first element of the list has index value

1, the second element has index value 2, etc., and the last has index value n . An *ordered pair* is a two-tuple.

An alternative, and equivalent, representation for an n -tuple $\langle q_1, q_2, \dots, q_n \rangle$ would be the sequence $\langle s_0, s_1, \dots, s_{n-1} \rangle$, where $\forall i \in \{1, 2, \dots, n\}, s_{i-1} = q_i$. It may also be noted that an element of a Cartesian product can be represented as a tuple. Thus, given $n \in \mathbb{N}$, an n -tuple drawn from some set A is a member of the Cartesian product A^n . (See Revisions and Changes Digest item 10 on page 66.)

Definition 25 (Subsequence): The sequence t is said to be a *subsequence of sequence* s if and only if $\exists r \subseteq s, \exists$ a sequence $q \in r^{\mathbb{N}} \ni$

1. $[(i, (m, a)), (i+1, (n, b))] \in q \Rightarrow$
 - (a) $m < n$ and
 - (b) $\neg [\exists (k, c) \in r \ni m < k < n]$ and
2. $t = \{(i, v) : \exists (m, v) \in r \ni (i, (m, v)) \in q\}$.

Definition 26 (Set of Sequences of Members of a Set Without Repeats): $\Xi^* : \wp(\Omega) \rightarrow \wp(\Omega^{\mathbb{N}}) \ni$

$$\forall Q \in \wp(\Omega), s \in \Xi^*(Q) \Leftrightarrow s \text{ is a sequence } \ni$$

1. $(\bullet, a) \in s \Rightarrow a \in Q$ and
2. $(i, a), (j, a) \in s \Rightarrow i = j$.

For each set Q , the function Ξ^* defines the set of all possible sequences of (not necessarily all) members of Q , without repeats, i.e., if $s \in \Xi^*(Q)$, then no member of Q appears twice in s .

Definition 27 (Set of Sequences of Members of a Subset of a Set Allowing Repeats): $\Xi^{**} : \wp(\Omega) \rightarrow \wp(\Omega^{\mathbb{N}}) \ni$

$$\forall Q \in \wp(\Omega), s \in \Xi^{**}(Q) \Rightarrow s \text{ is a sequence } \ni$$

$$[(\bullet, a) \in s \Rightarrow a \in Q].$$

Definition 28 (Set of Sequences of All Members of a Finite Set Without Repeats):

$$\Xi : \Omega_F \rightarrow \wp(\Omega^{\mathbb{N}}) \ni \forall Q \in \Omega_F, s \in \Xi(Q) \Leftrightarrow$$

$$s \text{ is a sequence } \ni$$

1. **len** $(s) = |Q|$ and
2. $a \in Q \Leftrightarrow \exists (\bullet, a) \in s$.

For each finite set Q , the function $\Xi(Q)$ defines the set of the sequences of all of the members of Q .

Definition 29 (Random member of a set):

$$\mathbf{rndmember} : \Omega_F \rightarrow \Omega \ni$$

$$\forall Q \in \Omega_F, \exists \xi \in \Xi(Q) \ni$$

$$\mathbf{rndmember}(Q) = \xi[\mathbf{rndint}(0, |Q| - 1)].$$

Given a finite set Q , **rndmember** (Q) returns a random member of Q .

Definition 30 (Random subset of finite set): $\mathbf{RND}: \mathbb{N}^+ \times \Omega_F \rightarrow \Omega_F \ni$

$$\left[Q \in \Omega_F, |Q| \geq n \in \mathbb{N}^+ \right] \Rightarrow$$

$$\mathbf{RND}(n, Q) = \mathbf{rdmember}(\{A \subseteq Q: |A| = n\}).$$

Given a non-empty finite set Q and a positive integer $n \leq |Q|$, $\mathbf{RND}(n, Q)$ returns a random subset of Q whose cardinality is n .

Definition 31 (Function to return the maximum value of a set of values):

$$\mathbf{max}: \wp(\mathbb{R}) \rightarrow \mathbb{R} \ni \forall \text{ closed set } Q \subset \mathbb{R}, \exists x \in Q \ni$$

1. $y \in Q \Rightarrow y \leq x$
2. $\mathbf{max}(Q) = x$

(See Revisions and Changes Digest item 11, page 66.)

The function \mathbf{max} returns the largest value in a set of numerical values.

Definition 32 (Function to return the minimum value of a set of values):

$$\mathbf{min}: \wp(\mathbb{R}) \rightarrow \mathbb{R} \ni \forall \text{ closed set } Q \subset \mathbb{R}, \exists x \in Q \ni$$

1. $y \in Q \Rightarrow y \geq x$
2. $\mathbf{min}(Q) = x$

(See Revisions and Changes Digest item 12, page 66.)

The function \mathbf{min} returns the least value in a set of numerical values.

Definition 33 (String): s is said to be a *string* if and only if $s \in \Xi^{**}(\{x: x \text{ is an ASCII character}\})$.

Definition 34 (\mathbf{S}): $\mathbf{S} = \{s: s \text{ is a string}\}$.

4.3. Domain-Specific Definitions

4.3.1. System Input Data

Definition 35 (Set of network-station IDs): $S_0 = \{s \in \mathbf{S}: s \text{ represents the ID of a station in either the Space Network or the Ground Network}\}$

The string ‘‘TDRS-B’’ is an example of a station ID. S_0 is supplied as input data to the scheduling system.

Definition 36 (Set of antenna IDs):

$$A^* = \{a \in \mathbf{S}: a \text{ represents an antenna ID}\}$$

A^* is supplied as input data to the scheduling system.

Definition 37 (Set of antenna-attribute tuples): $A_0 \subseteq [S_0 \times$

$$A^* \times \{\text{‘‘S’’}, \text{‘‘K’’}, \text{‘‘K1’’}, \text{‘‘K2’’}\} \times \{\text{‘‘MA’’}, \text{‘‘SA’’}\}] \ni$$

$$\left[a = (a_1, \dots, a_4) \in A_0, a_4 = \text{‘‘MA’’} \right] \Rightarrow a_3 = \text{‘‘S’’}$$

A_0 contains a 4-tuple for each antenna in the communications support infrastructure. The elements of each 4-tuple identify the basic antenna attributes (the station where the antenna is located, the antenna’s ID, the antenna’s frequency band, and the antenna’s signal service capability). A_0 is supplied as input data to the scheduling system.

Note: It is assumed that all SA antennas are able to support S band.

Definition 38 (Function to return the number of SA antennas in service at a given station): $S_0^{\text{SA}}: S_0 \rightarrow \mathbb{N} \ni s \in S_0 \Rightarrow$

$$S_0^{\text{SA}}(s) = \left| \{a \in A_0: a_1 = s, a_4 = \text{‘‘SA’’}\} \right|.$$

Given station s , $S_0^{\text{SA}}(s)$ returns the number of SA antennas in service at station s . This information is computed from input data for the scheduling system.

Definition 39 (Set of communications resource-outage intervals): $O \subseteq S_0 \times A^* \times \mathbb{N}^2 \ni (o_1, \dots, o_4) \in O \Rightarrow$

$$o_3 \text{ is a start time and } o_4 \text{ is an end time.}$$

O is the set of communications resource-outage intervals, each corresponding to times known in advance when data communications via prescribed antennas will be unavailable. O is supplied as input data to the scheduling system.

Definition 40 (Set of user IDs):

$$U_0 = \{u \in \mathbf{S}: u \text{ represents a user ID}\}$$

A user (see definition of User on page 18) is any system capable of communications via an antenna in NASA’s space data-communications infrastructure. U_0 is supplied as input data to the scheduling system.

Definition 41 (POCC Operation Periods): $P: U_0 \rightarrow \wp(\bar{\mathbb{Z}})$.

Given user u , $P(u)$ is the set of time intervals during which the user’s Project Operations Control Center (POCC) is in operation. The intervals belonging to the set $P(u)$ have start and end times specified as seconds of offset from some standard epoch. If a POCC is always in operation, then there is only one interval specified, the start time of which is the start of the scheduling period and the end time of which is an arbitrary, sufficiently large number. P is supplied as input data to the scheduling system.

Definition 42 (Set of all link IDs):

$$L^* = \{x \in \mathbf{S}: x \text{ represents a link ID}\}$$

L^* is provided as input data to the scheduling system

Definition 43 (Set of all mission event types): $M^* = \{x \in \mathbf{S}: x \text{ represents a mission event type}\}$

M^* is provided as input data to the scheduling system. An example of the set of mission event types would be: ‘‘NIL’’, ‘‘ORBIT’’, ‘‘COMM-VIEW-PERIOD’’, ‘‘IN-

VIEW”, “DAY-LIGHT”, “NIGHT”, “OVER-WATER”, “OVER-LAND”, “SUN-RISE”, “MOON-RISE”, “SUN-SET”, “MOON-SET”, “DAY”, “WEEK”, “MONTH”}. Some mission event types do not relate to a station in the support infrastructure. For example, for the mission event type “MOON-SET”, a station ID is irrelevant, and so, for a member of M (see Definition 46, page 22) for that mission event type, the value of μ_3 could be given as \bullet . Mission event type “NIL” is reserved for cases where a user-support requirement/request is specified in relation to an exact time interval (i.e., a “specific” requirement, as opposed to a “generic” requirement; see definition of User Requirement, page 18).

COMM-VIEW-PERIODs are assumed to be intervals during which RF communications with a given user via a given network station are possible. COMM-VIEW-PERIODs are determined in advance (and are assumed herein to be provided as input data), by considering all factors that affect communications performance, as computed, for example, by the NASA Communications Link Analysis and Simulation System (CLASS) [19] and the “Automated Conflict Resolution System” and the “TDRS Look Angle System” [20]. It is assumed that, according to the mission plan, the spacecraft attitude will be adjusted and maintained as needed to enable the appropriate on-board antenna(s) to receive and/or radiate signals from/to the designated support antenna.

Definition 44 (Users’ communications links information):
 $L_0 \subseteq U_0 \times L^* \times \{\text{“S”}, \text{“K”}, \text{“K1”}, \text{“K2”}\} \times$
 $\{\text{“MA”}, \text{“SA”}\} \times \{\text{“FWD”}, \text{“RTN”}\} \times \{\text{“RCP”}, \text{“LCP”}\} \times$
 $\mathbb{N}^+ \ni (\lambda_1, \dots, \lambda_7) \in L_0 \Rightarrow$

1. λ_4 indicates which type of signal service (Multiple Access or Single Access) is required,
2. λ_5 indicates the direction of data flow,
3. λ_6 indicates the signal polarization required, and
4. λ_7 represents a data rate in units of Kbps (i.e., 10^3 bits per second).

L_0 provides all relevant information about all users’ communications links. The schedule-generation algorithm requires this information in order to find a schedule that will satisfy requests for communications services. L_0 is supplied as input data to the scheduling system.

Definition 45 (Maximum allowed return data rate): **MAXALLOWEDRTNRATE** is a fixed parameter, provided as input data for a given scheduling scenario, applicable to the entire data-communications support infrastructure, defining the maximum allowed data rate (in units of Kbps) for all return-data communications links combined at any given instant.

Definition 46 (Set of mission event instances):

$$M \subseteq U_0 \times M^* \times S_0 \times \mathbb{N}^2 \ni (\mu_1, \dots, \mu_5) \in M \Rightarrow$$

1. μ_4 is a start time
2. μ_5 is an end time.

M , supplied as input data to the scheduling system, is the set of mission event instances. The preparation of M requires a computational resource such as CLASS.

Definition 47 (Set of communications links for given user):

$$L: U_0 \rightarrow \wp(L_0) \ni L(u) = \left\{ \lambda = (u, \lambda_2, \dots, \lambda_7) \in L_0 \right\}$$

For a given user u , $L(u)$ is u ’s set of communications links. The function L can be regarded as a table of data supplied as input to the scheduling system.

Definition 48: $V: S_0 \times U_0 \rightarrow \wp(M) \ni$

$$\left[(s, u) \in S_0 \times U_0, (\mu_1, \dots, \mu_5) \in V(s, u) \right] \Rightarrow$$

$$\left[\mu_1 = u, \mu_2 = \text{“COMM-VIEW-PERIOD”}, \mu_3 = s \right]$$

The function V , given $(s, u) \in S_0 \times U_0$, returns the set of all communications view periods for station s and user u . V , effectively a table of data, is supplied as input to the scheduling system.

Definition 49 (Potential Interference Intervals):

$$I: S_0^2 \times L_0^2 \rightarrow \wp(\mathbb{Z}) \ni$$

$$\left[(s, s', \lambda, \lambda') \in S_0^2 \times L_0^2, \zeta \in I(s, s', \lambda, \lambda') \right] \Leftrightarrow$$

$$\exists (\mu_1, \dots, \mu_5) \in V(s, \lambda_1), \exists (\mu'_1, \dots, \mu'_5) \in V(s', \lambda'_1) \ni$$

1. $\zeta \subseteq [\mu_4, \mu_5] \cap [\mu'_4, \mu'_5] \neq \emptyset$,
2. $[t \in \zeta, \text{links } \lambda \text{ and } \lambda' \text{ are active at time } t \text{ via stations } s \text{ and } s', \text{ respectively}] \Leftrightarrow \text{link } \lambda \text{ suffers unacceptable degradation due to interference by link } \lambda'$,
3. $\forall \beta \in \mathbb{Z} \ni \zeta \subset \beta, \exists t \in \beta \setminus \zeta \ni \text{at time } t$,
 - (a) links λ and λ' are active via stations s and s' , respectively, and
 - (b) link λ does not suffer unacceptable degradation due to interference by link λ' .

Potential interference intervals are supplied as input data by (for example) the NASA CLASS interference analysis system (IAS) (see Introduction, page 8). Note that CLASS can supply potential interference intervals for the cases where a user’s communications link would be degraded by RF energy from a non-specific source such as cellular-telephone signal emitters or other non-NASA sources such as radars. In such cases, the interfering “link” would have a CLASS-supplied link ID and user ID.

Definition 50 (Function to return NASA-assigned user priority):

$$\Phi_0: U_0 \rightarrow \mathbb{N} \ni \forall u \in U_0, \Phi_0(u) \text{ is the NASA-assigned mission-priority value } \ni$$

$$[u' \in U_0, u' \neq u, \text{ and } u' \text{ has lower priority than } u] \Rightarrow$$

$$\Phi_0(u') < \Phi_0(u).$$

NASA-assigned mission priorities are supplied to the scheduling system as input data.

Definition 51 (Normalized user priority mapping):

$$\begin{aligned} \Phi: U_0 \rightarrow \mathbb{R} \ni \\ \left[u \in U_0, m = \mathbf{max}(\{x: \exists u' \in U_0, x = \Phi_0(u')\}) \right] \Rightarrow \\ \Phi(u) = m^{-1}\Phi_0(u). \end{aligned}$$

Φ , supplied as input data to the scheduling system, maps NASA-assigned mission priorities to the interval $[0, 1]$.

Definition 52 (Service):

$$Y \subseteq L_0 \times \mathbb{N}^4 \ni (\lambda, s^-, s^+, d^-, d^+) \in Y \Rightarrow$$

1. s^- and s^+ represent, respectively, the minimum and maximum allowed start-time offset from some given reference time, and
2. d^- and d^+ represent, respectively, the minimum and maximum allowed duration of the service.

Y is the set of tuples $(\lambda, s^-, s^+, d^-, d^+)$ that specifies users' services in terms of links, earliest and latest start-time offsets, and minimum and maximum durations, and is supplied as input data to the scheduling system.

Definition 53 (User-Prescribed Prototype Event List):

$$\begin{aligned} C: U_0 \rightarrow \Xi^*(\Xi^*(Y)) \ni \\ \left[u \in U_0, k \in \mathbb{N}^+, k < \mathbf{len}(C(u)), \right. \\ \left. i \in \mathbb{N}, i < \mathbf{len}(C(u)[k]), \right. \\ \left. C(u)[k][i] = (\lambda, \bullet, \bullet, \bullet, \bullet) \right] \Rightarrow \lambda \in L(u). \end{aligned}$$

p is said to be a *prototype event* if and only if $\exists u \in U_0, \exists k \in \mathbb{N} \ni (k, p) \in C(u)$. (See Revisions and Changes Digest item 13, page 66.)

$C(u)$ is the user- u prescribed list (sequence) of prototype communications events for user u . Every communications event scheduled by the algorithm for the given user u will match the values of some element of $C(u)$, with leeway on the duration and the start-time offset relative to a given prototype event start time. The mission-operations project for each user u supplies the list $C(u)$ as input data to the scheduling system.

Definition 54 (User Requirements): R_0 is a set each element of which is an ordered 17-tuple $(r_1, \dots, r_{17}) \ni$

1. $r_1 \in \mathbf{S}$ represents a requirement ID,
2. $r_2 \in U_0$ is a string representing a user ID,
3. r_3 is a subsequence of the sequence $C(r_2)$, the list of prototype events prescribed by user r_2 ,
4. $r_4 \in M^*$ is a string representing a mission-event type for user r_2 ,

5. $r_5 \in \mathbb{N}$ is a mission-event skip factor specifying how many mission events of type r_4 must be skipped between consecutive instances of prototype events in the sequence r_3 ,
6. $r_6 \in \mathbb{Z}$ is seconds of offset of an instance of a prototype event in the sequence r_3 from the start (if $r_8 = \text{"START"}$), or end (if $r_8 = \text{"END"}$) of a mission event of type r_4 ,
7. $r_7 \in \mathbb{Z}$ is seconds of offset of an instance of a prototype event in the sequence r_3 from the start (if $r_9 = \text{"START"}$), or end (if $r_9 = \text{"END"}$) of a mission event of type r_4 ,
8. $r_8 \in \{\text{"START"}, \text{"END"}\}$ indicates whether the start of a prototype event instance is relative to the start or end of an instance of a mission event of type r_4 ,
9. $r_9 \in \{\text{"START"}, \text{"END"}\}$ indicates whether the end of a prototype event instance is relative to the start or end of an instance of a mission event of type r_4 ,
10. $r_{10} \in \mathbb{N}^+$ represents the total volume of data desired to be returned from the user spacecraft in units of 10^3 bits during any instance of a prototype event in the sequence r_3 ,
11. $r_{11} \in \{\text{"Y"}, \text{"N"}\}$ indicates whether the user's POCC must be open during an instance of a prototype event in the sequence r_3 , where "N" means the POCC is not required to be open,
12. $r_{12} \in \{\text{"Y"}, \text{"N"}\}$ indicates whether mutual interference may be neglected in scheduling communications events,
13. $r_{13} \in \mathbb{N}$ is the minimum allowed separation, in seconds, between any two consecutive instances of a prototype event in the sequence r_3 during any given communications event window as defined below,
14. $r_{14} \in \mathbb{N}$ is the maximum allowed separation, in seconds, between any two consecutive prototype event instances,
15. $r_{15} \in \mathbb{N}$ is the offset, in seconds, from the start of the scheduling period to the earliest time at which any instance of a prototype event in the sequence r_3 is allowed to start,
16. $r_{16} \in \mathbb{N}$ is the offset, in seconds, from the start of the scheduling period to the latest time at which any instance of a prototype event in the sequence r_3 is allowed to end, and
17. $r_{17} \in \mathbb{N}$ is the nominal prototype-event start time offset, in seconds, from the start of the scheduling period.

R_0 , given as input data by the users, is a set each element of which is a user requirement. A requirement specifies either repeating or singleton (nonrepeating) communications events to satisfy the user's needs for communications via stations in the network. A requirement may specify (via parameters r_{13} through r_{16} , and via the last four parameters in each of the user's defined services (see Definition 52 (page 23))) loose or tight tolerances on positioning of events in time.

Example of a communications event window relative to the mission event type "SUNRISE": Starting 20 sec before each third sunrise, ending 15 min after the sunrise. In this example, the mission-event skip-factor (r_5) would be 2.

For every requirement $r = (r_1, \dots, r_{17})$ where the mission event type is "NIL", there is a mission-event instance $\mu = (\mu_1, \dots, \mu_5)$ in M where $\mu_2 = \text{"NIL"}$ and $[\mu_4, \mu_5] = [r_{15}, r_{16}]$.

4.3.2. Scheduling-Algorithm-Specific Definitions

Definition 55 (Service Instantiation):

$$Y^I: Y \rightarrow \wp(\mathbb{N} \times A_0 \times \mathbb{N}^2) \ni \\ \forall y = (\lambda = (\lambda_1, \dots, \lambda_7), s^-, s^+, d^-, d^+) \in Y, \\ (t, a, s, d) \in Y^I(y) \Leftrightarrow$$

1. $a = (a_1, a_2, a_3 = \lambda_3, a_4 = \lambda_4) \in A_0$,
2. $s^- \leq s \leq s^+$, and
3. $d^- \leq d \leq d^+$

Y^I returns, for each defined service y , the set of all possible instantiations (t, a, s, d) of y where the link might be activated on the assigned antenna a during the interval $[(t + s), (t + s + d)]$.

Definition 56 (Function to return largest overlap of service with given view period):

$$Y^V: \mathbb{N} \times M \times Y \rightarrow \bar{\mathbb{Z}} \ni \\ \left[\left(t, \mu^V = (\mu_1^V, \dots, \mu_5^V), \right. \right. \\ \left. \left. y = (\lambda = (\lambda_1, \dots, \lambda_7), s^-, s^+, \bullet, d^+) \right) \in \right. \\ \left. \mathbb{N} \times M \times Y \ni \mu_1^V = \lambda_1, \mu^V \in V(\mu_3^V, \mu_1^V) \right] \Rightarrow \\ Y^V(t, \mu^V, y) = [\mu_4^V, \mu_5^V] \cap [(t + s^-), (t + s^+ + d^+)].$$

$Y^V(t, \mu^V, y)$ is the largest interval during which the service y instantiation, relative to the given offset t from the start of the scheduling period, would overlap the given view period μ^V . The interval $Y^V(t, \mu^V, y)$ covers any possible instantiation of y relative to the given view period μ^V and the given offset t from the start of the scheduling period.

Definition 57 (List of users):

$$U: R_0 \rightarrow U_0 \ni r = (r_1, \dots, r_{17}) \in R_0 \Rightarrow U(r) = r_2.$$

$U(r)$ represents the user for which r is a prescribed requirement.

Definition 58 (Time-ordered sequence of mission events for a given user requiring a given mission event type):

$$M_{\text{type}}: R_0 \rightarrow \Xi^*(M) \ni \\ \forall r = (r_1, \dots, r_{17}) \in R_0, \\ M_{\text{type}}(r) = \xi \in \Xi^*(M) \Leftrightarrow$$

1. $[i \in \mathbb{N}, i < \text{len}(\xi)] \Rightarrow \xi_i[1] = r_2 \text{ and } \xi_i[2] = r_4 \text{ and}$
2. $[i, j \in \mathbb{N}, i < j < \text{len}(\xi)] \Rightarrow \\ [\xi_i[4], \xi_i[5]] < [\xi_j[4], \xi_j[5]]$

(See Revisions and Changes Digest item 14, page 66.)

Given $r \in R_0$, $M_{\text{type}}(r)$ is the time-ordered sequence of all the members of M for user r_2 that have mission event type r_4 .

Definition 59 (Function to return a start time or an end time for a given mission event):

$$M_T: \{\text{"START"}, \text{"END"}\} \times M \rightarrow \mathbb{N} \ni \\ (x, \mu = (\mu_1, \dots, \mu_5)) \in \text{dom}(M_T) \Rightarrow$$

$$M_T(x, \mu) = \begin{cases} \mu_4 & \text{if } x = \text{"START"} \\ \mu_5 & \text{if } x = \text{"END"} \end{cases}$$

M_T returns a start time or an end time for a given mission event. The returned time is the reference time relative to which a prototype event will be instantiated according to r_6 .

Definition 60 (Function to return the reference time for a given mission event):

$$\text{tref}: R_0 \times M \rightarrow \mathbb{Z} \ni \\ \forall (r = (r_1, \dots, r_{17}), \\ \mu = (\mu_1, \dots, \mu_5)) \in R_0 \times M, \\ \text{tref}(r, \mu) = t_p \Leftrightarrow \\ \mu_1 = r_2, \mu_2 = r_4, \\ t_p = \begin{cases} r_{15} & \text{if } r_4 = \text{"NIL"} \\ M_T(r_8, \mu) + r_6 & \text{otherwise.} \end{cases}$$

(See Revisions and Changes Digest item 15, page 66.)

$\text{tref}(r, \mu)$ returns the reference time specified by requirement r relative to any mission event μ of type r_4 , with respect to which any prototype communications event would be scheduled, subject to the skip factor r_5 .

Definition 61 (Function to return indexes to mission event instances determined the skip factor):

$$M_{\text{skips}}: R_0 \times \mathbb{N} \rightarrow \Xi^*(\mathbb{N}) \ni \\ \forall (r = (r_1, \dots, r_{17}), i) \in R_0 \times \mathbb{N}, \\ M_{\text{skips}}(r, i) = \xi \in \Xi^*(\mathbb{N}) \Leftrightarrow$$

1. $i \leq r_5$,

2. $n = \text{len}(M_{\text{type}}(r)) \Rightarrow$
 $\text{len}(\xi) = \lceil (n - n \bmod (r_5 + 1)) / (r_5 + 1) \rceil$, and
3. $[j \in \mathbb{N}, j < \text{len}(\xi)] \Rightarrow \xi[j] = j(r_5 + 1) + i$

$M_{\text{skips}}(r, i)$ is the list of indexes into $M_{\text{type}}(r)$ such that, starting with the mission event instance $M_{\text{type}}(r)[i]$, these elements of $M_{\text{type}}(r)$ correspond to the mission event instances determined by applying the skip factor r_5 . The concept (see Figure 3 (page 25)) of applying a skip factor having the value n entails the process of (1) starting with some given mission event instance, (2) ignoring (i.e., skipping) the next n mission event instances, (3) keeping the next mission event instance, (4) skipping the next n mission event instances, etc. Note that in normal practice, the starting point for the process will be not some arbitrary member of $M_{\text{type}}(r)$, but instead will be $M_{\text{type}}(r)[0]$, i.e., the first mission event instance of type r_4 . This practice satisfies the normal mission expectation that in maximizing the satisfaction of mission requirements, no opportunities for enabling data communications will be foregone.

Definition 62 (Function to return a given user's POCC operation period that has the largest intersection with a given interval):

$$P_{\text{max}} : R_0 \times \bar{\mathbb{Z}} \rightarrow \bar{\mathbb{Z}} \ni$$

$$\left[(r, \beta) \in R_0 \times \bar{\mathbb{Z}}, P_{\text{max}}(r, \beta) = \zeta \in \bar{\mathbb{Z}} \right] \Leftrightarrow$$

1. $\zeta \in P(U(r))$,
2. $g = \zeta \cap \beta \neq \emptyset$, and
3. $[\eta \in P(U(r)), h = \eta \cap \beta] \Rightarrow g^+ - g^- \geq h^+ - h^-$

$P_{\text{max}}(r, \beta)$ returns the POCC operation period for user $U(r)$ that has the largest intersection with the given interval β .

Definition 63 (Function to return the largest interval in the given view period during which a given service can be instantiated):

$$Y_{\text{max}}^{\text{I}} : R_0 \times \mathbb{N}^2 \times M^2 \rightarrow \mathbb{N} \times A_0 \times \mathbb{N}^2 \ni$$

$$\forall (r = (r_1, \dots, r_{17}), k, n, \mu = (r_2, r_4, \mu_3, \mu_4, \mu_5),$$

$$\mu^{\text{V}} = (\mu_1^{\text{V}}, \dots, \mu_5^{\text{V}})) \in \text{dom}(Y_{\text{max}}^{\text{I}}),$$

$$Y_{\text{max}}^{\text{I}}(r, k, n, \mu, \mu^{\text{V}}) = (t_p, a, s, d) \Leftrightarrow$$

1. $(t_p, a = (a_1, \dots, a_4), s, d) \in \text{codomain}(Y_{\text{max}}^{\text{I}})$,
2. $\mu^{\text{V}} \in V(a_1, r_2)$,
3. $t_p = \text{tref}(r, \mu)$,
4. $k < \text{len}(r_3)$,
5. $n < \text{len}(r_3[k])$,
6. $(t_p, a, s, d) \in Y^{\text{I}}(r_3[k][n])$,
7. $[(o_1 = a_1, o_2 = a_2, a_3, o_4) \in O \Rightarrow$
 $[o_3, o_4] \cap [(t_p + s), (t_p + s + d)] = \emptyset]$,

Instances of mission event of a type specified by parameter r_4 :

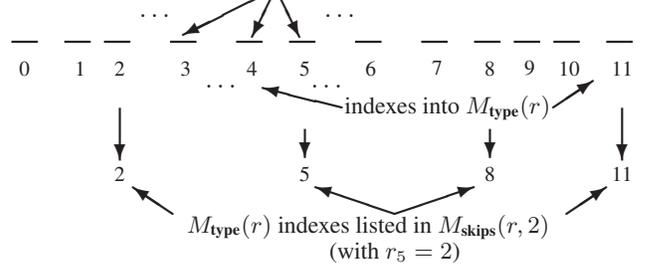


Figure 3. An example illustrating the skip-factor concept. Twelve instances of a mission event of type r_4 are shown along the time line. These instances have indexes 0 through 11 in the sequence $M_{\text{type}}(r)$. The relevant parameter values (see Definition 61 (page 24)) in this example are the skip factor, $r_5 = 2$, and the index, $i = 2$, of the first (i.e., the left-most) instance of a mission event of type r_4 where an instance of a prototype event is to be scheduled. Thus, after skipping the next two instances of a mission event of type r_4 , the next index in the list is $M_{\text{skips}}(r, 2)(1) = 5$. Note, however, that in normal practice, the starting point for this process will be not some arbitrary member of $M_{\text{type}}(r)$, but instead will be $M_{\text{type}}(r)[0]$, i.e., the first mission-event instance of type r_4 , corresponding to setting the index i to 0.

8. $\left[\left[r_4 = \text{"NIL"} \right], [\mu_4, \mu_5] = [r_{15}, r_{16}], \right.$
 $\beta = Y^{\text{V}}(t_p, \mu^{\text{V}}, r_3[k][n]), \beta \neq \emptyset,$
 $\left[[r_{11} = \text{"N"} \right], \eta = \beta] \vee$
 $\left[r_{11} = \text{"Y"} \right], \zeta = P_{\text{max}}(r, \beta), \eta = \zeta \cap \beta] \vee$
 $\left[r_4 \neq \text{"NIL"} \right], \zeta = [t_p, \mu_5^{\text{V}}],$
 $\beta = \zeta \cap Y^{\text{V}}(t_p, \mu^{\text{V}}, r_3[k][n]), \beta \neq \emptyset,$
 $\left[[r_{11} = \text{"N"} \right], \eta = \beta] \vee$
 $\left[r_{11} = \text{"Y"} \right], \zeta = P_{\text{max}}(r, \beta), \eta = \zeta \cap \beta] \right] \vee$
9. $t_p + s = \eta^-, d = \eta^+ - (t_p + s) > 0$

$Y_{\text{max}}^{\text{I}}(r, k, n, \mu, \mu^{\text{V}})$ gives the largest interval in the given view period μ^{V} where, (a) with respect to $[r_{15}, r_{16}]$ (when

r_4 = “NIL”) or (b) with respect to the given mission event μ (when the given requirement specifies the start of the given prototype event in relation to the given mission event), the given service (stipulated by (r, k, n)) can be instantiated with an antenna assigned avoiding any resource outage.

Definition 64 (Function to return the set of all possible instantiations of the given prototype event for a given mission event type with antenna assignments avoiding resource outage intervals):

$$C_0^{\text{PRM}}: \mathbb{N} \times R_0 \times M \rightarrow \wp(\Xi^*(\text{codomain}(Y^{\text{I}}))) \ni \\ \left[(k, r = (r_1, \dots, r_{17}), \mu = (\mu_1 = r_2, \mu_2 = r_4, \dots, \mu_5)) \right. \\ \left. \in \text{dom}(C_0^{\text{PRM}}), \right. \\ \left. p \in C_0^{\text{PRM}}(k, r, \mu) \right] \Leftrightarrow$$

1. $k < \text{len}(r_3), \text{len}(p) = \text{len}(r_3[k]),$
2. $\left[\left[t_p = \text{tref}(r, \mu), n \in \mathbb{N}, n < \text{len}(p) \right] \Rightarrow \right.$
 - (a) $\exists (t_p, a, s, d) \in Y^{\text{I}}(r_3[k][n]) \ni$
 $p[n] = (t_p, a, s, d),$
 - (b) $\exists \mu^{\text{V}} = (\mu_1^{\text{V}}, \dots, \mu_5^{\text{V}}) \in M, \exists [s^*, d^*] \in \bar{\mathbb{Z}},$
 $\exists (t_p, a, s^*, d^*) = Y_{\text{max}}^{\text{I}}(k, r, n, \mu, \mu^{\text{V}}) \ni$
 $[s, (s + d)] \subseteq [s^*, (s^* + d^*)].$

$C_0^{\text{PRM}}(k, r, \mu)$ is the set of all possible instantiations of the k th prototype event in the list r_3 for the mission event μ of type r_4 , with antenna assignments avoiding resource outage intervals.

Definition 65 (Set of All Possible Schedules):

$$\Theta \subseteq \text{codomain}(C_0^{\text{PRM}}) \ni \forall \theta \in \Theta, \\ \left[(k, r = (r_1, \dots, r_{17}), \mu = (\mu_1 = r_2, \mu_2 = r_4, \dots, \mu_5)) \right. \\ \left. \in \mathbb{N} \times R_0 \times M, \right. \\ \left. k < \text{len}(r_3), \right. \\ \left. x \in C_0^{\text{PRM}}(k, r, \mu), y \in C_0^{\text{PRM}}(k, r, \mu), \right. \\ \left. x \in \theta, \text{ and } y \in \theta \right] \Rightarrow x = y$$

Θ is the set of all possible schedules.

A schedule is a set of prototype-event instantiations, with no more than one such instantiation for each instance of the mission event type stipulated by each requirement.

For each $(k, r = (r_1, \dots, r_{17}), \mu = (r_2, r_4, \dots, \mu_5)) \in \mathbb{N} \times R_0 \times M$, with $k < \text{len}(r_3)$, the *scheduling objective* is to schedule an instance of the prototype event $r_3[k]$ so as to transmit a total quantity of data equal to $r_{10} \times 10^3$ bits, subject to

- the minimum and maximum communications-event separations r_{13} and r_{14} and
- the mission-event skip factor r_5 .

Definition 66 (Function to return the degree to which a given schedule satisfies a given requirement’s skip factor):

$$\text{skipsat}^{\text{R}^*}: \Theta \times R_0 \rightarrow \mathbb{R} \ni \\ \left[(\theta, r = (r_1, \dots, r_{17})) \in \Theta \times R_0, \right. \\ \left. n = \text{len}(M_{\text{skips}}(r, 0)), \right. \\ \left. Q = \left\{ p: \exists j, k \in \mathbb{N} \ni j < n, k < \text{len}(r_3), \right. \right. \\ \left. \left. p \in C_0^{\text{PRM}}(k, r, M_{\text{type}}(r)) \left[M_{\text{skips}}(r, 0)[j] \right] \right\}, \right. \\ \left. p \in \theta \right\} \Rightarrow$$

$$\text{skipsat}^{\text{R}^*}(\theta, r) = \begin{cases} n/|Q| & \text{if } |Q| > 0 \\ 1000 & \text{if } |Q| = 0. \end{cases}$$

(Note that the value 1000 is arbitrary, chosen to severely reduce the fitness score of θ when the set Q is empty.)

Given a schedule θ and a requirement r , $\text{skipsat}^{\text{R}^*}$ returns a value representing the ratio of the number of elements in $M_{\text{skips}}(r, 0)$ to the number of prototype events scheduled for the members indexed by $M_{\text{skips}}(r, 0)$. This final value will be exactly 1 if the skip factor requirement is satisfied (the possibility that prototype event instances will be scheduled for other mission events is irrelevant for this metric), and will be a larger value otherwise. The assumption is that, from the start of the scheduling period, the first mission event of type r_4 will have a mandatory first prototype-event instantiation, then r_5 mission events of type r_4 will be skipped, and then the next mission event of type r_4 will have a mandatory prototype-event instantiation, with this pattern repeated for the remainder of the scheduling period.

(See Revisions and Changes Digest item 16, page 66, giving an equivalent but simpler formulation.)

Definition 67 (Function to return the degree to which a given schedule satisfies the skip factor for all requirements):

$$\text{violations}^{\text{SKIP}^*}: \Theta \rightarrow \mathbb{R} \ni \\ \theta \in \Theta \Rightarrow \text{violations}^{\text{SKIP}^*}(\theta) = \\ |R_0|^{-1} \sum_{r \in R_0} \text{skipsat}^{\text{R}^*}(\theta, r)$$

Given a schedule θ , $\text{violations}^{\text{SKIP}^*}$ returns the total of the metrics for all requirements as to how well their skip factors are satisfied—averaged over all requirements. For a perfect schedule, this metric will be exactly 1, and will be a larger value otherwise.

Definition 68: $\text{skip}^{\text{FILL-R}^*}: \Theta \times R_0 \rightarrow \mathbb{R} \ni$

$$\left[(\theta, r = (r_1, \dots, r_{17})) \in \Theta \times R_0, N = \text{len}(M_{\text{type}}(r)), \right. \\ \left. h = \text{len}(M_{\text{type}}(r)) - \text{len}(M_{\text{skips}}(r, 0)), \right. \\ \left. Q = \left\{ p: \exists m, k \in \mathbb{N}, k < \text{len}(r_3), \right. \right. \\ \left. \left. m < \text{len}(M_{\text{type}}(r)), \neg m \in M_{\text{skips}}(r, 0), \right. \right.$$

$$\left. p \in C_0^{\text{PRM}}(k, r, M_{\text{type}(r)}[m]), p \in \theta \right\} \Rightarrow$$

$$\text{skip}^{\text{FILL-R}^*}(\theta, r) = 1 + h^{-1}|Q|$$

Given a schedule θ and a requirement r , $\text{skip}^{\text{FILL-R}^*}$ returns 1 plus the ratio of $|Q|$ (the number of prototype-event instantiations that are not required under the mission-event skip requirement r_5 for mission events of type r_4), to h (the number of mission event instances of type r_4 that are required to be skipped). This metric has the value 1 if the schedule is perfect (i.e., there are no prototype events instantiated when not required), and has a greater value otherwise. See the statement of the assumption under Definition 66. (Note that $h = 0$ corresponds to an impossible condition, namely, that all of the instances of the mission event of type r_4 are to be skipped.)

Definition 69: violations^{SKIPFILL*} : $\Theta \rightarrow \mathbb{N} \ni$

$$\theta \in \Theta \Rightarrow \text{violations}^{\text{SKIPFILL}^*}(\theta) =$$

$$|R_0|^{-1} \sum_{r \in R_0} \text{skip}^{\text{FILL-R}^*}(\theta, r)$$

Given a schedule θ , $\text{violations}^{\text{SKIPFILL}^*}$ returns the total count, for all requirements r , of prototype-event instantiations that are not required under the mission-event skip requirement r_5 for mission events of type r_4 —averaged over all requirements.

Definition 70: start^P : $\text{codomain}(C_0^{\text{PRM}}) \rightarrow \mathbb{N} \ni$

1. $p \in \text{codomain}(C_0^{\text{PRM}}) \Rightarrow \exists(k, r, \mu) \in \mathbb{N} \times R_0 \times M \ni$
 $p \in C_0^{\text{PRM}}(k, r, \mu),$
2. $\left[Q = \{v : \exists(t, \bullet, s, \bullet) \in p \ni v = t + s\} \right] \Rightarrow$
 $\text{start}^{\text{P}}(p) = \min(Q)$

Given the instantiation p of a prototype event, $\text{start}^{\text{P}}(p)$ returns the earliest start time of any service instantiation in the event.

Definition 71: end^P : $\text{codomain}(C_0^{\text{PRM}}) \rightarrow \mathbb{N} \ni$

1. $p \in \text{codomain}(C_0^{\text{PRM}}) \Rightarrow \exists(k, r, \mu) \in \mathbb{N} \times R_0 \times M \ni$
 $p \in C_0^{\text{PRM}}(k, r, \mu),$
2. $\left[Q = \{v : \exists(t, \bullet, s, d) \in p \ni v = t + s + d\} \right] \Rightarrow$
 $\text{end}^{\text{P}}(p) = \max(Q)$

Given the instantiation p of a prototype event, $\text{end}^{\text{P}}(p)$ returns the latest end time of any service instantiation in the event.

Definition 72: minsepsat^{P*} : $\Theta \times R_0 \rightarrow \mathbb{N} \ni$

$$\left[(\theta, r = (r_1, \dots, r_{17})) \in \Theta \times R_0, \right.$$

$$Q = \{ (p, p') \in \theta \times \theta : \exists(k, r, \mu) \in \mathbb{N} \times R_0 \times M \ni$$

$$p \in C_0^{\text{PRM}}(k, r, \mu),$$

$$\exists(k, r, \mu') \in \mathbb{N} \times R_0 \times M \ni p' \in C_0^{\text{PRM}}(k, r, \mu'),$$

$$\text{start}^{\text{P}}(p') > \text{end}^{\text{P}}(p) \}$$

$$\left. \text{start}^{\text{P}}(p') - \text{end}^{\text{P}}(p) < r_{13} \right\} \Rightarrow$$

$$\text{minsepsat}^{\text{P}^*}(\theta, r) = |Q|$$

Given a schedule θ and a requirement r , $\text{minsepsat}^{\text{P}^*}(\theta, r)$ returns the total count of pairs of prototype-event instantiations for requirement r in schedule θ that are separated by less than the minimum allowed separation r_{13} .

Definition 73: maxsepsat^{P*} : $\Theta \times R_0 \rightarrow \mathbb{N} \ni$

$$\left[(\theta, r = (r_1, \dots, r_{17})) \in \Theta \times R_0, \right.$$

$$Q = \{ (p, p') \in \theta \times \theta : \exists(k, r, \mu) \in \mathbb{N} \times R_0 \times M \ni$$

$$p \in C_0^{\text{PRM}}(k, r, \mu),$$

$$\exists(k, r, \mu') \in \mathbb{N} \times R_0 \times M \ni p' \in C_0^{\text{PRM}}(k, r, \mu'),$$

$$\text{start}^{\text{P}}(p') > \text{end}^{\text{P}}(p)$$

$$\neg \exists p'' \in \theta \ni \text{start}^{\text{P}}(p) < \text{start}^{\text{P}}(p'') < \text{start}^{\text{P}}(p')$$

$$\text{start}^{\text{P}}(p') - \text{end}^{\text{P}}(p) > r_{14} \}$$

$$\Rightarrow$$

$$\text{maxsepsat}^{\text{P}^*}(\theta, r) = |Q|$$

Given a schedule θ and a requirement r , $\text{maxsepsat}^{\text{P}^*}(\theta, r)$ returns the total count of pairs of consecutive prototype-event instantiations for requirement r in schedule θ that are separated by more than the maximum allowed separation r_{14} .

Definition 74: violations^{MINSEP*} : $\Theta \rightarrow \mathbb{R} \ni$

$$\theta \in \Theta \Rightarrow \text{violations}^{\text{MINSEP}^*}(\theta) =$$

$$1 + |R_0|^{-1} \sum_{r \in R_0} \text{minsepsat}^{\text{P}^*}(\theta, r)$$

Given a schedule θ , $\text{violations}^{\text{MINSEP}^*}(\theta)$ returns the value 1 plus the ratio, averaged over all requirements r , of the number of pairs of prototype-event instantiations for requirement r in schedule θ that are separated by less than the minimum allowed separation r_{13} to the number of elements (prototype-event instantiations) in the schedule. This metric will be exactly 1 for a perfect schedule and a greater value otherwise.

Definition 75: violations^{MAXSEP*} : $\Theta \rightarrow \mathbb{R} \ni$

$$\theta \in \Theta \Rightarrow \text{violations}^{\text{MAXSEP}^*}(\theta) =$$

$$1 + |R_0|^{-1} \sum_{r \in R_0} \text{maxsepsat}^{\text{P}^*}(\theta, r)$$

Given a schedule θ , $\text{violations}^{\text{MAXSEP}^*}(\theta)$ returns a value equal to 1 plus the ratio, averaged over all requirements r , of the number of pairs of prototype-event instantiations for requirement r in schedule θ that are separated by more than the minimum allowed separation r_{14} to the number of elements (prototype-event instantiations) in the schedule. This metric will be exactly 1 for a perfect schedule and a greater value otherwise.

Definition 76: schedolpairs : $\Theta \rightarrow \text{codomain}(C_0^{\text{PRM}})^2 \ni$

$$\theta \in \Theta \Rightarrow$$

$$\text{schedolpairs}(\theta) = \{ (p, p') \in \theta \times \theta :$$

$$\text{start}^P(p) < \text{start}^P(p') < \text{end}^P(p) \Big\}$$

Given a schedule θ , $\text{schedolpairs}(\theta)$ returns a set of overlapping pairs of members of θ so that not both (p, p') and (p', p) belong to the set and (p, p) does not belong to the set.

Definition 77: $\text{interf}^* : \Theta \rightarrow \mathbb{R} \ni \forall \theta \in \Theta$,

$$\left[\begin{aligned} Q &= \left\{ x : r = (r_1, \dots, r_{17}), \right. \\ &\quad r' = (r'_1, \dots, r'_{17}) \in R_0, k, k' \in \mathbb{N}, \\ &\quad \mu \text{ is an element of the sequence } M_{\text{type}}(r), \\ &\quad \mu' \text{ is an element of the sequence } M_{\text{type}}(r'), \\ &\quad k < \text{len}(r_3), k' < \text{len}(r'_3), \\ &\quad p \in C_0^{\text{PRM}}(k, r, \mu), p' \in C_0^{\text{PRM}}(k', r', \mu'), \\ &\quad (p, p') \in \text{schedolpairs}(\theta), \\ &\quad n, n' \in \mathbb{N}, n < \text{len}(r_3[k]), n' < \text{len}(r'_3[k']), \\ &\quad \lambda = (r_2, \lambda_2, \dots, \lambda_7), \lambda' = (r'_2, \lambda'_2, \dots, \lambda'_7) \in L_0, \\ &\quad (\lambda, \bullet, \bullet, \bullet, \bullet) = r_3[k][n], \\ &\quad (\lambda', \bullet, \bullet, \bullet, \bullet) = r'_3[k'][n'], \\ &\quad p[n] = (t, a = (a_1, \dots, a_4), s, d), \\ &\quad p'[n'] = (t', a' = (a'_1, \dots, a'_4), s', d'), \\ &\quad e \in I(a_1, a'_1, \lambda, \lambda'), \\ &\quad e \cap [t + s, t + s + d] \cap [t' + s', t' + s' + d'] \neq \emptyset, \\ &\quad \left. x = (p, p', n, n') \right\} \Rightarrow \\ \text{interf}^*(\theta) &= 1 + |\theta|^{-1} |Q| \end{aligned} \right.$$

interf^* returns the value 1 plus an integer representing the instances where interference exists between two active links in a pair of prototype-event instantiations in the schedule, averaged over all elements (prototype-event instantiations) in the schedule. This metric will be exactly 1 for a perfect schedule and a greater value otherwise.

Definition 78: $\text{endpts} : \Theta \rightarrow \wp(\mathbb{N}) \ni \theta \in \Theta \Rightarrow$

$$\text{endpts}(\theta) = \left\{ x : r = (r_1, \dots, r_{17}) \in R_0, \right. \\ \left. \begin{aligned} k &\in \mathbb{N}, k < \text{len}(r_3), \\ (\bullet, \mu) &\in M_{\text{type}}(r), \\ p &\in C_0^{\text{PRM}}(k, r, \mu), p \in \theta, (t, a, s, d) \in p, \\ [x = t + s \vee x = t + s + d] &\end{aligned} \right\}$$

The function $\text{endpts}(\theta)$ returns the set of all of the endpoints of all service instantiations in all prototype-event instantiations in schedule θ .

Definition 79: $\text{endpts}_{\text{seq}} : \Theta \rightarrow \Xi^*(\mathbb{N}) \ni \forall \theta \in \Theta$,

$$\begin{aligned} \xi &\in \text{endpts}(\theta)_{\text{seq}} \Leftrightarrow \\ \xi &\in \Xi^*(\text{endpts}(\theta)) \text{ and} \\ [i &\in \mathbb{N}, i + 1 < \text{len}(\xi) \Rightarrow \xi[i] < \xi[i + 1]] \end{aligned}$$

The function $\text{endpts}_{\text{seq}}(\theta)$ converts the set $\text{endpts}(\theta)$ into an increasing sequence of times on the timeline.

Case	$\kappa(a, c, d)$
(1) $c = \text{"SA"}$ and $d = \bullet$	2
(2) $c = \text{"MA"}$ and $d = \text{"FWD"}$	1
(3) $c = \text{"MA"}$ and $d = \text{"RTN"}$	5

Table 1. Space Network Forward and Return Link constraints from the Space Network Users' Guide (SNUG)

Case	$\kappa'(a, c, d)$
(1) $c = \text{"SA"}$ and $d = \bullet$	4
(2) $c = \text{"MA"}$ and $d = \text{"FWD"}$	2
(3) $c = \text{"MA"}$ and $d = \text{"RTN"}$	20

Table 2. Ground Network Forward and Return Link constraints

Definition 80: $\text{resourceusage} : \Theta \times \mathbb{N} \rightarrow$

$$\begin{aligned} &\wp(\mathbb{N} \times R_0 \times M \times \mathbb{N}^3 \times L_0 \times A_0) \ni \\ &\left[\begin{aligned} \theta &\in \Theta, i \in \mathbb{N}, \\ i + 1 &< |\text{endpts}(\theta)|, \\ (k, r = (r_1, \dots, r_{17}), \mu) &\in \mathbb{N} \times R_0 \times M, \\ p &\in C_0^{\text{PRM}}(k, r, \mu), p \in \theta, \\ n &\in \mathbb{N}, n < \text{len}(r_3[k]), \\ r_3[k][n] &= (\lambda = (r_2, \lambda_2, \dots, \lambda_7), \bullet, \bullet, \bullet, \bullet), \\ (t, a, s, d) &\in \mathbb{N} \times A_0 \times \mathbb{N}^2, \\ p[n] &= (t, a, s, d), \\ \zeta &= [\text{endpts}_{\text{seq}}(\theta)[i], \text{endpts}_{\text{seq}}(\theta)[i + 1]], \\ \zeta \cap [(t + s), (t + s + d)] &\neq \emptyset \Rightarrow \\ (k, r, \mu, n, \zeta^-, \zeta^+, \lambda, a) &\in \text{resourceusage}(\theta, i) \end{aligned} \right. \end{aligned}$$

Given a schedule θ and an index i into the list of endpoints of all the service instantiations in θ , $\text{resourceusage}(\theta, i)$ returns a set of 8-tuples containing values representing the resources used during the interval starting at the time $\text{endpts}_{\text{seq}}[i]$.

Definition 81: $\kappa^{\text{SN}} : \{\text{"S"}, \text{"K"}, \text{"K1"}, \text{"K2"}\} \times$
 $\{\text{"MA"}, \text{"SA"}\} \times \{\text{"FWD"}, \text{"RTN"}\} \rightarrow \mathbb{N} \ni$
 $(b, c, d) \in \{\text{"S"}, \text{"K"}, \text{"K1"}, \text{"K2"}\} \times \{\text{"MA"}, \text{"SA"}\} \times$
 $\{\text{"FWD"}, \text{"RTN"}\} \Rightarrow$
 $\kappa^{\text{SN}}(b, c, d) = 0$, except as shown in Table 1

κ^{SN} returns the constraints on combinations of Space Network resource usage in any schedule.

Table 1 states the station constraints that are provided as input to the scheduling system. For example, for any TDRS, there can be only one MAF, only five MAR, only two SSAF, only two SSAR, only two KSAF, and only two KSAR[19].

Definition 82:

$$\begin{aligned} \kappa^{\text{GN}}: \{“S”, “K”, “K1”, “K2”\} \times \{“MA”, “SA”\} \times \\ \{“FWD”, “RTN”\} \rightarrow \mathbb{N} \ni \\ (b, c, d) \in \{“S”, “K”, “K1”, “K2”\} \times \{“MA”, “SA”\} \times \\ \{“FWD”, “RTN”\} \Rightarrow \\ \kappa^{\text{GN}}(b, c, d) = 0, \text{ except as shown in Table 2} \end{aligned}$$

κ^{GN} returns the constraints on combinations of Ground Network resource usage in any schedule.

Table 2 states the ground-terminal constraints that are provided as input to the scheduling system. For example, for WSC (White Sands Complex), there can be only two MAF, only 20 MAR, only four SSAF, only four SSAR, only four KSAF, and only four KSAR. However, this is a simplification that would have to be dealt with, both in a more realistic formulation of the SN scheduling problem and in a full specification of the problem solution (i.e., as a schedule-generation algorithm to be implemented in a fielded, production-level scheduling system). The actual SA constraints are subject to additional rules that likewise would need to be included in the specification for a fielded scheduling system. For example, in the SNUG, Note 5 in Figure 3-1 “Telecommunications Services for each SGLT” [19] states the following:

The SN can simultaneously support S-band and K-band (either Ku or Ka for TDRS spacecraft F8 through F10) forward and/or return services through one SA antenna to the same ephemeris.

The present disclosure is based on a formulation of the NASA data-communications scheduling problem that does not embody the above distinction (or any other distinction) between TDRS spacecraft in the Space Network. Special cases and changes in infrastructure constraints are generally expected over time and must be reflected in timely updates to any operational scheduling system. In this sense, the present disclosure should therefore be considered to represent an approach and a method that can be adapted to the actual data-communications scheduling problem.

Definition 83: violations_{B-C-D}^{SN-ENDPTS}:

$$\begin{aligned} \Theta \times \mathbb{N} \times \{“S”, “K”, “K1”, “K2”\} \times \\ \{“MA”, “SA”\} \times \{“FWD”, “RTN”\} \rightarrow \mathbb{N} \ni \\ \left[\theta \in \Theta, i \in \mathbb{N}, i + 1 < |\text{endpts}(\theta)|, \right. \\ (b, c, d) \in \{“S”, “K”, “K1”, “K2”\} \times \\ \{“MA”, “SA”\} \times \{“FWD”, “RTN”\}, \\ Q = \left\{ x: x = (\bullet, \bullet, \bullet, \bullet, \bullet, \bullet, \lambda = (\lambda_1, \dots, \lambda_7), \bullet) \in \right. \\ \left. \text{resourceusage}(\theta, i), \right. \\ \left. \lambda_3 = b, \lambda_4 = c, \lambda_5 = d, \right\}, \\ v_{\text{SN}} = |Q| - \kappa^{\text{SN}}(b, c, d) \Rightarrow \\ \text{violations}_{\text{B-C-D}}^{\text{SN-ENDPTS}}(\theta, i, b, c, d) = \max(\{0, v_{\text{SN}}\}) \end{aligned}$$

The function $\text{violations}_{\text{B-C-D}}^{\text{SN-ENDPTS}}(\theta, i, b, c, d)$ returns the count of violations of the constraints on usage of Space Network resource (b, c, d) in the interval i in schedule θ .

Definition 84: violations_{B-C-D}^{GN-ENDPTS}:

$$\begin{aligned} \Theta \times \mathbb{N} \times \{“S”, “K”, “K1”, “K2”\} \times \\ \{“MA”, “SA”\} \times \{“FWD”, “RTN”\} \rightarrow \mathbb{N} \ni \\ \left[\theta \in \Theta, i \in \mathbb{N}, i + 1 < |\text{endpts}(\theta)|, \right. \\ (b, c, d) \in \{“S”, “K”, “K1”, “K2”\} \times \\ \{“MA”, “SA”\} \times \{“FWD”, “RTN”\}, \\ Q = \left\{ x: x = (\bullet, \bullet, \bullet, \bullet, \bullet, \bullet, \lambda = (\lambda_1, \dots, \lambda_7), \bullet) \in \right. \\ \left. \text{resourceusage}(\theta, i), \right. \\ \left. \lambda_3 = b, \lambda_4 = c, \lambda_5 = d, \right\}, \\ v_{\text{GN}} = |Q| - \kappa^{\text{GN}}(b, c, d) \Rightarrow \\ \text{violations}_{\text{B-C-D}}^{\text{GN-ENDPTS}}(\theta, i, b, c, d) = \max(\{0, v_{\text{GN}}\}) \end{aligned}$$

The function $\text{violations}_{\text{B-C-D}}^{\text{GN-ENDPTS}}(\theta, i, b, c, d)$ returns the count of violations of the constraints on usage of Ground Network resource (b, c, d) in the interval i in schedule θ .

Definition 85: violations^{SN-ENDPTS}: $\Theta \times \mathbb{N} \rightarrow \mathbb{R} \ni$

$$\begin{aligned} \left[(\theta, i) \in \Theta \times \mathbb{N}, h = |\text{endpts}(\theta)| - 1 \right] \Rightarrow \\ \text{violations}^{\text{SN-ENDPTS}}(\theta, i) = \\ h^{-1} \sum_{(b, c, d) \in \text{dom}(\kappa^{\text{SN}})} \text{violations}_{\text{B-C-D}}^{\text{SN-ENDPTS}}(\theta, i, b, c, d) \end{aligned}$$

The function $\text{violations}^{\text{SN-ENDPTS}}(\theta, i)$ returns the count of violations of the constraints on usage of all Space Network resources in the interval i in schedule θ , averaged by the number of elements in $\text{endpts}(\theta)$ less 1.

Definition 86: violations^{GN-ENDPTS}: $\Theta \times \mathbb{N} \rightarrow \mathbb{R} \ni$

$$\begin{aligned} \left[(\theta, i) \in \Theta \times \mathbb{N}, h = |\text{endpts}(\theta)| - 1 \right] \Rightarrow \\ \text{violations}^{\text{GN-ENDPTS}}(\theta, i) = \\ h^{-1} \sum_{(b, c, d) \in \text{dom}(\kappa^{\text{GN}})} \text{violations}_{\text{B-C-D}}^{\text{GN-ENDPTS}}(\theta, i, b, c, d) \end{aligned}$$

$\text{violations}^{\text{GN-ENDPTS}}(\theta, i)$ returns the count of violations of the constraints on usage of all Ground Network resources in the interval i in schedule θ , averaged by the number of elements in $\text{endpts}(\theta)$ less 1.

Definition 87: violations^{SN}: $\Theta \rightarrow \mathbb{R} \ni$

$$\begin{aligned} \theta \in \Theta \Rightarrow \text{violations}^{\text{SN}}(\theta) = \\ 1 + |\theta|^{-1} \sum_{\substack{i \in \mathbb{N} \\ i+1 < |\text{endpts}(\theta)|}} \text{violations}^{\text{SN-ENDPTS}}(\theta, i) \end{aligned}$$

$\text{violations}^{\text{SN}}(\theta)$ returns a value equal to 1 plus the count of violations of the constraints on usage of all Space Network resources in schedule θ , averaged over all elements (prototype-event instantiations) in the schedule. This metric

will be exactly 1 for a perfect schedule and a greater value otherwise.

Definition 88: violations^{GN}: $\Theta \rightarrow \mathbb{R} \ni$
 $\theta \in \Theta \Rightarrow \text{violations}^{\text{GN}}(\theta) =$
 $1 + |\theta|^{-1} \sum_{\substack{i \in \mathbb{N} \\ i+1 < |\text{endpts}(\theta)|}} \text{violations}^{\text{GN-ENDPTS}}(\theta, i)$

violations^{GN}(θ) returns a value equal to 1 plus the count of violations of the constraints on usage of all Ground Network resources in schedule θ , averaged over all elements (prototype-event instantiations) in the schedule. This metric will be exactly 1 for a perfect schedule and a greater value otherwise.

Definition 89:

usage^{STATION-SA-ENDPTS}: $\Theta \times \mathbb{N} \times S_0 \rightarrow \mathbb{N} \ni$
 $\left[(\theta, i, s) \in \Theta \times \mathbb{N} \times S_0, i+1 < |\text{endpts}(\theta)|, \right.$
 $Q = \left\{ x : x = (\bullet, \bullet, \bullet, \bullet, \bullet, \bullet, \lambda = (\lambda_1, \dots, \lambda_7), \right.$
 $a = (a_1, \dots, a_4) \in \text{resourceusage}(\theta, i),$
 $\left. \left. a_1 = s, \lambda_4 = a_4 = \text{“SA”} \right\} \Rightarrow$
usage^{STATION-SA-ENDPTS}(θ, i, s) = $|Q|$

Given a schedule θ , given an index i into the sequence of endpoints in $\text{endpts}_{\text{seq}}(\theta)$, and given a station s , **usage^{STATION-SA-ENDPTS}(θ, i, s)** returns the demand for SA antenna support on s .

Definition 90:

violations^{STATION-SA-ENDPTS}: $\Theta \times \mathbb{N} \times S_0 \rightarrow \mathbb{N} \ni$
 $\left[(\theta, i, s) \in \Theta \times \mathbb{N} \times S_0, \right.$
 $v_{\text{SA}} = \text{usage}^{\text{STATION-SA-ENDPTS}}(\theta, i, s) - S_0^{\text{SA}}(s) \Rightarrow$
violations^{STATION-SA-ENDPTS}(θ, i, s) = $\max(\{0, v_{\text{SA}}\})$

violations^{STATION-SA-ENDPTS}(θ, i, s) returns the count of violations of the constraints on usage of SA antennas on station s in the i th interval in schedule θ .

Definition 91: violations^{SA-ENDPTS}: $\Theta \times \mathbb{N} \rightarrow \mathbb{R} \ni$

$\left[(\theta, i) \in \Theta \times \mathbb{N}, h = |\text{endpts}(\theta)| - 1 \right] \Rightarrow$
violations^{SA-ENDPTS}(θ, i) =
 $h^{-1} \sum_{s \in S_0} \text{violations}^{\text{STATION-SA-ENDPTS}}(\theta, i, s)$

violations^{SA-ENDPTS}(θ, i) returns the count of violations of the constraints on usage of SA antennas in the i th interval in schedule θ , averaged by the total number of elements in $\text{endpts}(\theta)$ less 1.

Definition 92: violations^{SA}: $\Theta \rightarrow \mathbb{R} \ni$

$\theta \in \Theta \Rightarrow \text{violations}^{\text{SA}}(\theta) =$

$$1 + |\theta|^{-1} \sum_{\substack{i \in \mathbb{N} \\ i+1 < |\text{endpts}(\theta)|}} \text{violations}^{\text{SA-ENDPTS}}(\theta, i)$$

violations^{SA}(θ) returns a value equal to 1 plus the count of violations of the constraints on usage of SA antennas in schedule θ , averaged over all elements (prototype-event instantiations) in the schedule. This metric will be exactly 1 for a perfect schedule and a greater value otherwise.

Definition 93: stnsw^{PEI}: $\Theta \times \mathbb{N} \times R_0 \times M \rightarrow \mathbb{Z}^+ \ni$

$\forall (\theta, k, r = (r_1, \dots, r_{17}), \mu) \in \Theta \times \mathbb{N} \times R_0 \times M,$
if $p \in C_0^{\text{PRM}}(k, r, \mu),$
if $p \in \theta,$ and

$Q = \left\{ x : \exists i, j \in \mathbb{N}, \exists \lambda \in L_0 \ni$

$i, j < \text{len}(r_3[k]), i \neq j,$

$r_3[k][i] = (\lambda, \bullet, \bullet, \bullet, \bullet),$

$r_3[k][j] = (\lambda, \bullet, \bullet, \bullet, \bullet),$

$p[i] = (t, a = (a_1, \dots, a_4), s, d),$

$p[j] = (t, a' = (a'_1, \dots, a'_4), s', d'),$

$s + d \leq s',$

$a_1 \neq a'_1,$

$[m \in \mathbb{N}, m < \text{len}(r_3[k]),$

$m \neq i, m \neq j,$

$p[m] = (t, a^* = (a_1^*, \dots, a_4^*), s^*, d^*),$

$r_3[k][m] = (\lambda, \bullet, \bullet, \bullet, \bullet) \Rightarrow s' \leq s^*,$ and

$x = (i, j, \lambda) \left. \right\},$ then

stnsw^{PEI}(θ, k, r, μ) = $|Q|$

stnsw^{PEI}(θ, k, r, μ) returns the number of station switches that occur in the prototype-event instantiation p identified by (k, r, μ) in schedule θ .

In this disclosure, for the metric **stnsw^{PEI}**, a station switch is said to occur if, for a prototype-event instantiation p identified by $(k, r = (r_1, \dots, r_{17}), \mu)$, there are two services $r_3[k][i] = (\lambda, \bullet, \bullet, \bullet, \bullet)$ and $r_3[k][j] = (\lambda, \bullet, \bullet, \bullet, \bullet), i, j \in \mathbb{N}, i, j < \text{len}(r_3[k]), i \neq j$ such that if $p[i] = (t, a = (a_1, \dots, a_4), s, d)$ and $p[j] = (t, a' = (a'_1, \dots, a'_4), s', d')$, and $s + d \leq s'$, then $a_1 \neq a'_1$ (i.e., the station providing the link service changes from the earlier service instantiation to the later), and if $m \in \mathbb{N}, m < \text{len}(r_3[k]), m \neq i, m \neq j, p[m] = (t, a^* = (a_1^*, \dots, a_4^*), s^*, d^*),$ and $r_3[k][m] = (\lambda, \bullet, \bullet, \bullet, \bullet)$, then $s' \leq s^*$. Other possible definitions of “station switch” may be substituted for the one given above or may be included as additional metrics.

Definition 94: violations^{STNSW}: $\Theta \rightarrow \mathbb{R} \ni$

$\theta \in \Theta \Rightarrow \text{violations}^{\text{STNSW}}(\theta) =$
 $1 + |\theta|^{-1} \sum_{\substack{r=(r_1, \dots, r_{17}) \in R_0 \\ k \in \mathbb{N} \ni k < \text{len}(r_3) \\ (\bullet, \mu) \in M_{\text{type}}(r)}} \text{stnsw}^{\text{PEI}}(\theta, k, r, \mu)$

violations^{STNSW}(θ) returns a value equal to 1 plus the number of station switches that occur totaled for all prototype-

event instantiations in schedule θ , averaged over all elements (prototype-event instantiations) in the schedule. This metric will be exactly 1 for a perfect schedule and a greater value otherwise.

Definition 95: $\text{rtndatarate}^{\text{COMBINED}}: \Theta \times \mathbb{N} \rightarrow \mathbb{N} \ni$

$$\left[(\theta, i) \in \Theta \times \mathbb{N}, i + 1 < |\text{endpts}(\theta)| \right] \Rightarrow \\ \text{rtndatarate}^{\text{COMBINED}}(\theta, i) = \sum_{\substack{(\bullet, \bullet, \bullet) \in \text{resourceusage}(\theta, i) \\ \lambda_5 = \text{"RTN"}}} \lambda_7$$

$\text{rtndatarate}^{\text{COMBINED}}(\theta, i)$ returns, for the interval indexed by i in schedule θ , the combined data rate in all the active “RTN” links.

Definition 96: $\text{violation}^{\text{RTNRATE}}: \Theta \times \mathbb{N} \rightarrow \mathbb{N} \ni$

$$\left[(\theta, i) \in \Theta \times \mathbb{N}, i + 1 < |\text{endpts}(\theta)|, \right. \\ \left. x = \text{rtndatarate}^{\text{COMBINED}}(\theta, i), \right. \\ \left. [x > \text{MAXALLOWEDRTNRATE} \Rightarrow v = 1], \right. \\ \left. [x \leq \text{MAXALLOWEDRTNRATE} \Rightarrow v = 0] \right] \Rightarrow \\ \text{violation}^{\text{RTNRATE}}(\theta, i) = v$$

Given a schedule θ and an index i into the sequence of endpoints in $\text{endpts}_{\text{seq}}(\theta)$, $\text{violation}^{\text{RTNRATE}}(\theta, i)$ returns the value 1 if the total of the data-rate values in all of the active “RTN” links during the interval in schedule θ whose left endpoint is indexed by i exceeds the value of the fixed parameter MAXALLOWEDRTNRATE , and returns 0 otherwise.

Definition 97: $\text{violations}^{\text{RTNRATE}}: \Theta \rightarrow \mathbb{R} \ni$

$$\theta \in \Theta, h = |\text{endpts}(\theta)| - 1 \Rightarrow \\ \text{violations}^{\text{RTNRATE}}(\theta) = 1 + h^{-1} \sum_{\substack{i \in \mathbb{N} \\ i+1 < |\text{endpts}(\theta)|}} \text{violation}^{\text{RTNRATE}}(\theta, i)$$

$\text{violations}^{\text{RTNRATE}}(\theta)$ returns a value equal to 1 plus the number of intervals in schedule θ in which a data-rate violation exists, averaged over the total number of intervals in schedule θ . This metric will be exactly 1 for a perfect schedule and a greater value otherwise.

Definition 98: $\text{satisfied}^{\text{PEI}}: \Theta \times \mathbb{N} \times R_0 \times M \rightarrow \mathbb{R} \ni$

$$\left[(\theta, k, r = (r_1, \dots, r_{17}), \mu) \in \Theta \times \mathbb{N} \times R_0 \times M, \right. \\ \left. p \in C_0^{\text{PRM}}(k, r, \mu), p \in \theta, x \in \mathbb{R}, \right. \\ \left. Q = \left\{ (d, e) \in \mathbb{N}^2: n \in \mathbb{N}, \right. \right. \\ \left. \left. n < \text{len}(r_3[k]), (\bullet, \bullet, \bullet, d) = p[n], \right. \right. \\ \left. \left. (\lambda = (\lambda_1, \dots, \lambda_7), \bullet, \bullet, \bullet, \bullet) = r_3[k][n], \right. \right. \\ \left. \left. \lambda_5 = \text{"RTN"}, e = \lambda_7 \right\}, \right. \\ \left. x = \sum_{(d,e) \in Q} ed > 0 \right] \Rightarrow \\ \text{satisfied}^{\text{PEI}}(\theta, k, r, \mu) = 1 - |1 - x/r_{10}|$$

$\text{satisfied}^{\text{PEI}}(\theta, k, r, \mu)$ returns the total data bits returned to the POCC during the prototype-event instantiation identified by (k, r, μ) in schedule θ , divided by the desired volume r_{10} of data returned in the instantiation of any prototype event scheduled to satisfy r . This metric will be exactly 1 when the total number of returned data bits equals the desired quantity, and will be a nonnegative number less than 1 otherwise.

Definition 99: $\text{satisfied}^{\text{R}}: \Theta \times R_0 \rightarrow \mathbb{R} \ni$

$$\left[(\theta, r = (r_1, \dots, r_{17})) \in \Theta \times R_0, \right. \\ \left. Q = \left\{ p: \exists (k, \mu) \in \mathbb{N} \times M \ni \right. \right. \\ \left. \left. p \in C_0^{\text{PRM}}(k, r, \mu), p \in \theta \right\} \right. \\ \left. h = \max(\{1, |Q|\}) \right] \Rightarrow \\ \text{satisfied}^{\text{R}}(\theta, r) = h^{-1} \sum_{\substack{k \in \mathbb{N}, k < \text{len}(r_3) \\ (\bullet, \mu) \in M_{\text{type}}(r)}} \text{satisfied}^{\text{PEI}}(\theta, k, r, \mu)$$

$\text{satisfied}^{\text{R}}(\theta, r)$ returns, for requirement r , the ratio representing the satisfaction of the requirement r_{10} for total data bits returned by all the prototype-event instantiations for requirement r in schedule θ , averaged over all such prototype-event instantiations. The value returned is a nonnegative number not exceeding 1. The metric will have the value 1 if the schedule is perfect.

Definition 100: $\text{satisfied}^*: \Theta \rightarrow \mathbb{R} \ni \theta \in \Theta \Rightarrow$

$$\text{satisfied}^*(\theta) = 2 - \prod_{r \in R_0} \Phi(U(r)) \text{satisfied}^{\text{R}}(\theta, r)$$

$\text{satisfied}^*(\theta)$ returns, for all requirements r , a value equal to 2 minus the product of all of the user-priority-weighted ratios representing the satisfaction of the data-return requirements r_{10} for total data bits returned to Earth by all the prototype-event instantiations for requirement r in schedule θ . This metric corresponds to the overall degree to which the schedule satisfies all data-return requirements. The value returned will be exactly 1 for a perfect schedule and a greater value otherwise.

Definition 101: $\forall j \in \{0, \dots, 11\}, \mathbf{J}_j: \Theta \rightarrow \mathbb{R} \ni \theta \in \Theta \Rightarrow$

$$\begin{aligned} \mathbf{J}_0(\theta) &= \text{violations}^{\text{STNSW}}(\theta), \\ \mathbf{J}_1(\theta) &= \text{violations}^{\text{SKIP}^*}(\theta), \\ \mathbf{J}_2(\theta) &= \text{violations}^{\text{SKIPFILL}^*}(\theta), \\ \mathbf{J}_3(\theta) &= \text{violations}^{\text{MINSEP}^*}(\theta), \\ \mathbf{J}_4(\theta) &= \text{violations}^{\text{MAXSEP}^*}(\theta), \\ \mathbf{J}_5(\theta) &= \text{violations}^{\text{SN}}(\theta), \\ \mathbf{J}_6(\theta) &= \text{violations}^{\text{GN}}(\theta), \\ \mathbf{J}_7(\theta) &= \text{violations}^{\text{SA}}(\theta), \\ \mathbf{J}_8(\theta) &= \text{violations}^{\text{STNSW}}(\theta), \\ \mathbf{J}_9(\theta) &= \text{violations}^{\text{RTNRATE}}(\theta), \\ \mathbf{J}_{10}(\theta) &= \text{interf}^*(\theta), \text{ and} \\ \mathbf{J}_{11}(\theta) &= \text{satisfied}^*(\theta) \end{aligned}$$

Definition 102: fitness: $\Theta \rightarrow \mathbb{R} \ni \theta \in \Theta \Rightarrow$

$$\text{fitness}(\theta) = \prod_{j \in \{0, \dots, 11\}} \mathbf{J}_j(\theta)$$

This is the “fitness function”, which returns 1 for a perfect schedule and larger values for schedules that are not so good.

Note the perhaps unexpected numerical aspect of the fitness function defined above, by which a better schedule has a lower numerical value than a worse schedule. The value of the metric is unity for a perfect schedule, or a larger value for a less-than-perfect schedule.

We now define a series of functions that provide the genetic mutation and crossover transformations needed to evolve the working population during the operation of the schedule-generation algorithm (see Section 5 (page 35)).

Definition 103: rndpei: $\Theta \rightarrow \mathbb{N} \times R_0 \times M \ni$

$$\left[\begin{aligned} &\theta \in \Theta, r = \text{rndmember}(R_0), \\ &j = \text{rndint}(0, \text{len}(M_{\text{type}(r)}) - 1), \mu = M_{\text{type}(r)}[j], \\ &k = \text{rndint}(0, \text{len}(r_3) - 1), \\ &p \in C_0^{\text{PRM}}(k, r, \mu), p \in \theta \end{aligned} \right] \Rightarrow$$

$$\text{rndpei}(\theta) = (k, r, \mu)$$

Given a schedule θ , $\text{rndpei}(\theta)$ returns a parameter tuple (k, r, μ) that corresponds randomly to a prototype-event instantiation belonging to θ . This is a pseudo-function.

Definition 104: rndsvc: $\Theta \rightarrow \mathbb{N} \times R_0 \times M \times \mathbb{N} \ni$

$$\left[\begin{aligned} &\theta \in \Theta, (k, r, \mu) = \text{rndpei}(\theta), \\ &n = \text{rndint}(0, \text{len}(r_3[k]) - 1) \end{aligned} \right] \Rightarrow$$

$$\text{rndsvc}(\theta) = (k, r, \mu, n)$$

Given a schedule θ , $\text{rndsvc}(\theta)$ returns a parameter tuple (k, r, μ, n) that corresponds randomly to a service in a prototype-event instantiation belonging to θ . This is a pseudo-function.

Definition 105:

$$\text{modsvc: } \Theta \times \mathbb{N} \times R_0 \times M \times \mathbb{N}^2 \times A_0 \times \mathbb{N}^2 \rightarrow \Theta \ni$$

$$\left[\begin{aligned} &(\theta, k, r, \mu, n, t, a, s, d) \in \\ &\Theta \times \mathbb{N} \times R_0 \times M \times \mathbb{N}^2 \times A_0 \times \mathbb{N}^2, \\ &\theta' \in \Theta, \\ &p \in C_0^{\text{PRM}}(k, r, \mu), p \in \theta, \\ &p' \in C_0^{\text{PRM}}(k, r, \mu), p' \in \theta', \\ &\theta \setminus \{p\} = \theta' \setminus \{p'\}, \\ &[j \in \mathbb{N}, j < \text{len}(p), j \neq n] \Rightarrow p[j] = p'[j], \text{ and} \\ &p'[n] = (t, a, s, d) \end{aligned} \right] \Rightarrow$$

$$\text{modsvc}(\theta, k, r, \mu, n, t, a, s, d) = \theta'$$

Given the tuple $(\theta, k, r, \mu, n, t, a, s, d)$, the function modsvc returns a schedule identical to θ except with the service instantiation indexed by n in a prototype-event instantiation be-

longing to θ (and identified by the tuple (k, r, μ)) replaced with a service instantiation (t, a, s, d) .

Definition 106: slipsvc: $\Theta \rightarrow \Theta \ni$

$$\left[\begin{aligned} &(\theta, k, r, \mu, n, t, a, s_{\text{new}}, d) \in \\ &\Theta \times \mathbb{N} \times R_0 \times M \times \mathbb{N}^2 \times A_0 \times \mathbb{N}^2, \\ &(k, r, \mu, n) = \text{rndsvc}(\theta), \\ &p \in C_0^{\text{PRM}}(k, r, \mu), p \in \theta, p[n] = (t, a, s, d), \\ &\mu^{\mathbf{V}} = (\mu_1^{\mathbf{V}}, \dots, \mu_5^{\mathbf{V}}) \in M, \\ &\mu^{\mathbf{V}} \in V(a_1, r_2), \\ &(s^*, d^*) \in \mathbb{N}^2, \\ &(t, a, s^*, d^*) = Y_{\text{max}}^{\mathbf{I}}(r, k, n, \mu, \mu^{\mathbf{V}}), \text{ and} \\ &s_{\text{new}} = \text{rndint}(s^*, (s^* + d^* - d)) \end{aligned} \right] \Rightarrow$$

$$\text{slipsvc}(\theta) = \text{modsvc}(\theta, k, r, \mu, n, t, a, s_{\text{new}}, d)$$

Given a schedule θ , $\text{slipsvc}(\theta)$ returns a schedule identical to θ except with a randomly selected service instantiation in a prototype-event instantiation belonging to θ replaced with a service instantiation resulting from slipping the original service instantiation to the left or right by an allowed random amount. This is a pseudo-function.

Definition 107: chngsvcdur: $\Theta \rightarrow \Theta \ni$

$$\left[\begin{aligned} &(\theta, k, r, \mu) \in \Theta \times \mathbb{N} \times R_0 \times M, \\ &(k, r, \mu, n) = \text{rndsvc}(\theta), \\ &p \in C_0^{\text{PRM}}(k, r, \mu), p \in \theta, p[n] = (t, a, s, d), \\ &\mu^{\mathbf{V}} = (\mu_1^{\mathbf{V}}, \dots, \mu_5^{\mathbf{V}}) \in M, \\ &\mu^{\mathbf{V}} \in V(a_1, r_2), \\ &(s^*, d^*) \in \mathbb{N}^2, \\ &(t, a, s^*, d^*) = Y_{\text{max}}^{\mathbf{I}}(r, k, n, \mu, \mu^{\mathbf{V}}), \\ &(\lambda, \bullet, \bullet, d^-, d^+) \in L_0 \times \mathbb{N}^4, \\ &r_3[k][n] = (\lambda, \bullet, \bullet, d^-, d^+), \\ &d_{\text{max}} = \min(\{d^*, d^+\}), \text{ and} \\ &d_{\text{new}} = \text{rndint}(d^-, d_{\text{max}}) \end{aligned} \right] \Rightarrow$$

$$\text{chngsvcdur}(\theta) = \text{modsvc}(\theta, k, r, \mu, n, t, a, s, d_{\text{new}})$$

Given a schedule θ , $\text{chngsvcdur}(\theta)$ returns a schedule identical to θ except with a randomly selected service instantiation in a prototype-event instantiation belonging to θ replaced with a service instantiation resulting from changing the duration of the original service instantiation by an allowed random amount. This is a pseudo-function.

Definition 108: chngsvesta: $\Theta \rightarrow \Theta \ni$

$$\left[\begin{aligned} &(\theta, k, r, \mu) \in \Theta \times \mathbb{N} \times R_0 \times M, \\ &(k, r, \mu, n) = \text{rndsvc}(\theta), \\ &p \in C_0^{\text{PRM}}(k, r, \mu), p \in \theta, p[n] = (t, a, s, d), \\ &\xi \in \Xi(A_0), i = \text{rndint}(0, |A_0| - 1), \\ &j \in \mathbb{N}, j < \text{len}(\xi), \xi[j] = a, i \neq j, \\ &a' = \xi[i], a'_1 \neq a_1, a'_3 = a_3, a'_4 = a_4, \\ &\mu^{\mathbf{V}} = (\mu_1^{\mathbf{V}}, \dots, \mu_5^{\mathbf{V}}) \in M, \\ &\mu^{\mathbf{V}} \in V(a_1, r_2), \end{aligned} \right]$$

$$\begin{aligned} & (s^*, d^*) \in \mathbb{N}^2, \\ & (t, a', s^*, d^*) = Y_{\max}^{\mathbf{I}}(r, k, n, \mu, \mu^{\mathbf{V}}), \text{ and} \\ & [(t + s), (t + s + d)] \subseteq [(t + s^*), (t + s^* + d^*)] \Rightarrow \\ & \mathbf{chngsvcsta}(\theta) = \mathbf{modsvc}(\theta, k, r, \mu, n, t, a', s, d) \end{aligned}$$

Given a schedule θ , $\mathbf{chngsvcsta}(\theta)$ returns a schedule identical to θ except with a randomly selected service instantiation in a prototype-event instantiation belonging to θ replaced with a service instantiation resulting from changing the support antenna of the original service instantiation to a randomly selected allowed antenna on a different station. This is a pseudo-function.

Definition 109: $\mathbf{chngsvcant}: \Theta \rightarrow \Theta \ni$

$$\begin{aligned} & [(\theta, k, r, \mu) \in \Theta \times \mathbb{N} \times R_0 \times M, \\ & (k, r, \mu, n) = \mathbf{randsvc}(\theta), \\ & p \in C_0^{\mathbf{PRM}}(k, r, \mu), p \in \theta, p[n] = (t, a, s, d), \\ & \xi \in \Xi(A_0), i = \mathbf{randint}(0, |A_0| - 1), \\ & j \in \mathbb{N}, j < \mathbf{len}(\xi), \xi[j] = a, i \neq j, \\ & a' = \xi[i], a'_1 = a_1, a'_3 = a_3, a'_4 = a_4] \Rightarrow \\ & \mathbf{chngsvcant}(\theta) = \mathbf{modsvc}(\theta, k, r, \mu, n, t, a', s, d) \end{aligned}$$

Given a schedule θ , $\mathbf{chngsvcant}(\theta)$ returns a schedule identical to θ except with a randomly selected service instantiation in a prototype-event instantiation belonging to θ replaced with a service instantiation resulting from changing the support antenna of the original service instantiation to a randomly selected allowed antenna on the same station. This is a pseudo-function.

Definition 110: $\mathbf{replacepei}: \Theta \rightarrow \Theta \ni$

$$\begin{aligned} & [(\theta, \theta', k, k', r, \mu) \in \Theta^2 \times \mathbb{N}^2 \times R_0 \times M, \\ & (k, r, \mu) = \mathbf{randpei}(\theta), \\ & k' = \mathbf{randint}(0, \mathbf{len}(r_3) - 1), k' \neq k, \\ & p \in C_0^{\mathbf{PRM}}(k, r, \mu), p \in \theta, \\ & p' \in C_0^{\mathbf{PRM}}(k', r, \mu), p' \in \theta', \\ & \theta \setminus \{p\} = \theta' \setminus \{p'\}] \Rightarrow \\ & \mathbf{replacepei}(\theta) = \theta' \end{aligned}$$

Given a schedule θ , $\mathbf{replacepei}(\theta)$ returns a schedule identical to θ except a randomly selected prototype-event instantiation belonging to θ is replaced with an instantiation of a randomly selected different prototype event for the same requirement and for the same mission event relative to which the original prototype event was instantiated. This is a pseudo-function.

Definition 111: $\mathbf{cutexcesspei}: \Theta \rightarrow \Theta \ni$

$$\begin{aligned} & [(\theta, \theta', k, k', r, \mu) \in \Theta^2 \times \mathbb{N}^2 \times R_0 \times M, \\ & \mathbf{violations}^{\mathbf{SKIPFILL}^*}(\theta) > 0, \\ & Q = \{p: (k, i, r, \mu) \in \mathbb{N}^2 \times R_0 \times M, \end{aligned}$$

$$\begin{aligned} & i < \mathbf{len}(M_{\mathbf{type}}(r)), \neg i \in M_{\mathbf{skips}}(r, 0), \\ & \mu = M_{\mathbf{type}}(r)[i], p \in C_0^{\mathbf{PRM}}(k, r, \mu), p \in \theta \} \Rightarrow \\ & \mathbf{cutexcesspei}(\theta) = \theta \setminus Q \end{aligned}$$

Given a schedule θ , $\mathbf{cutexcesspei}(\theta)$ returns a schedule identical to θ except all excess prototype-event instantiations belonging to θ are excised. Excess prototype-event instantiations are those that cause the function $\mathbf{violations}^{\mathbf{SKIPFILL}^*}$ (see Definition 69) to return a value greater than 0.

Definition 112: $\mathbf{swappei}: \Theta^2 \rightarrow \Theta^2 \ni$

$$\begin{aligned} & [(\theta, \theta', e, e', k, k', r, \mu) \in \Theta^4 \times \mathbb{N}^2 \times R_0 \times M, \\ & \theta \neq \theta', (k, r, \mu) = \mathbf{randpei}(\theta), \\ & k' = \mathbf{randint}(0, \mathbf{len}(r_3) - 1), k' \neq k, \\ & p \in C_0^{\mathbf{PRM}}(k, r, \mu), p \in \theta, \\ & p' \in C_0^{\mathbf{PRM}}(k', r, \mu), p' \in \theta', \\ & e = \theta \setminus \{p\} \cup \{p'\}, \\ & e' = \theta' \setminus \{p'\} \cup \{p\}] \Rightarrow \\ & \mathbf{swappei}(\theta, \theta') = (e, e') \end{aligned}$$

Given a pair (θ, θ') of schedules, $\mathbf{swappei}(\theta, \theta')$ returns a pair (e, e') of schedules identical to (θ, θ') except a randomly selected prototype-event instantiation belonging to θ is swapped in θ' with an instantiation of a randomly selected different prototype event for the same requirement and for the same mission event relative to which the original prototype event was instantiated. This is a pseudo-function.

Definition 113: $\mathbf{swappeionr}: \Theta^2 \rightarrow \Theta^2 \ni$

$$\begin{aligned} & [(\theta, \theta', e, e') \in \Theta^4, \theta \neq \theta', \\ & r = \mathbf{randmember}(R_0), \\ & Q = \{x: x \in \theta, (k, r, \mu) \in \mathbb{N} \times R_0 \times M, \\ & \quad x \in C_0^{\mathbf{PRM}}(k, r, \mu)\} \ni \\ & [(k, r, \mu) \in \mathbb{N} \times R_0 \times M, x \in C_0^{\mathbf{PRM}}(k, r, \mu), \\ & \quad y \in C_0^{\mathbf{PRM}}(k, r, \mu), x \in Q, y \in Q] \Rightarrow x = y, \\ & B = \{x: x \in \theta', (k, r, \mu) \in \mathbb{N} \times R_0 \times M, \\ & \quad x \in C_0^{\mathbf{PRM}}(k, r, \mu)\} \ni \\ & [(k, r, \mu) \in \mathbb{N} \times R_0 \times M, x \in C_0^{\mathbf{PRM}}(k, r, \mu), \\ & \quad y \in C_0^{\mathbf{PRM}}(k, r, \mu), x \in B, y \in B] \Rightarrow x = y, \\ & e = (\theta \setminus Q) \cup B, \\ & e' = (\theta' \setminus B) \cup Q] \Rightarrow \\ & \mathbf{swappeionr}(\theta, \theta') = (e, e') \end{aligned}$$

Given a schedule pair (θ, θ') , $\mathbf{swappeionr}$ returns a schedule pair (e, e') identical to (θ, θ') except that for a randomly selected requirement r all prototype-event instantiations for r

belonging to θ are swapped with all prototype-event instantiations for r belonging to θ' .

Definition 114: swappeionu: $\Theta^2 \rightarrow \Theta^2 \ni$

$$\begin{aligned} & \left[(\theta, \theta', e, e') \in \Theta^4, \theta \neq \theta', \right. \\ & u = \mathbf{rdmember}(U_0), \\ & Q = \left\{ x: (k, r = (r_1, \dots, r_{17}), \mu) \in \mathbb{N} \times R_0 \times M, \right. \\ & \quad \left. r_2 = u, x \in C_0^{\text{PRM}}(k, r, \mu), x \in \theta \right\} \ni \\ & \left[(k, r, \mu) \in \mathbb{N} \times R_0 \times M, x \in C_0^{\text{PRM}}(k, r, \mu), \right. \\ & \quad \left. y \in C_0^{\text{PRM}}(k, r, \mu), x \in Q, y \in Q \right] \Rightarrow x = y, \\ & B = \left\{ x: (k, r = (r_1, \dots, r_{17}), \mu) \in \mathbb{N} \times R_0 \times M, \right. \\ & \quad \left. r_2 = u, x \in C_0^{\text{PRM}}(k, r, \mu), x \in \theta' \right\} \ni \\ & \left[(k, r, \mu) \in \mathbb{N} \times R_0 \times M, x \in C_0^{\text{PRM}}(k, r, \mu), \right. \\ & \quad \left. y \in C_0^{\text{PRM}}(k, r, \mu), x \in B, y \in B \right] \Rightarrow x = y, \\ & e = (\theta \setminus Q) \cup B, \\ & e' = (\theta' \setminus B) \cup Q \Big] \Rightarrow \\ & \mathbf{swappeionu}(\theta, \theta') = (e, e') \end{aligned}$$

Given a schedule pair (θ, θ') , **swappeionu** returns a schedule pair (e, e') identical to (θ, θ') except that for a randomly selected user u all prototype-event instantiations for u belonging to θ are swapped with all prototype-event instantiations for u belonging to θ' . (See Revisions and Changes Digest item 17, page 66.)

Definition 115: swapearlypeionr: $\Theta^2 \rightarrow \Theta^2 \ni$

$$\begin{aligned} & \left[(\theta, \theta', e, e') \in \Theta^4, \theta \neq \theta', \right. \\ & r = \mathbf{rdmember}(R_0), \mathbf{len}(M_{\text{type}}(r)) > 1, \\ & j = \mathbf{rdint}(0, \mathbf{len}(M_{\text{type}}(r)) - 2), \\ & Q = \left\{ x: x \in \theta, \right. \\ & \quad \left. (i, k, r = (r_1, \dots, r_{17}), \mu) \in \mathbb{N}^2 \times R_0 \times M, \right. \\ & \quad \left. i \leq j, \mu = M_{\text{type}}(r)[i], x \in C_0^{\text{PRM}}(k, r, \mu) \right\} \ni \\ & \left[(k, r, \mu) \in \mathbb{N} \times R_0 \times M, x \in C_0^{\text{PRM}}(k, r, \mu), \right. \\ & \quad \left. y \in C_0^{\text{PRM}}(k, r, \mu), x \in Q, y \in Q \right] \Rightarrow x = y, \\ & B = \left\{ x: x \in \theta' \right. \\ & \quad \left. (i, k, r = (r_1, \dots, r_{17}), \mu) \in \mathbb{N}^2 \times R_0 \times M, \right. \\ & \quad \left. i \leq j, \mu = M_{\text{type}}(r)[i], x \in C_0^{\text{PRM}}(k, r, \mu) \right\} \ni \\ & \left[(k, r, \mu) \in \mathbb{N} \times R_0 \times M, x \in C_0^{\text{PRM}}(k, r, \mu), \right. \\ & \quad \left. y \in C_0^{\text{PRM}}(k, r, \mu), x \in B, y \in B \right] \Rightarrow x = y, \\ & e = (\theta \setminus Q) \cup B, \\ & e' = (\theta' \setminus B) \cup Q \Big] \Rightarrow \end{aligned}$$

swapearlypeionr $(\theta, \theta') = (e, e')$

Given a schedule pair (θ, θ') , **swapearlypeionr** returns a schedule pair (e, e') identical to (θ, θ') except that for a randomly selected requirement r and a randomly selected mission event instance μ of type r_4 all prototype-event instantiations for r not later than μ belonging to θ are swapped with all prototype-event instantiations for r not later than μ belonging to θ' . (See Revisions and Changes Digest item 18, page 66.)

Definition 116: swapmidpeionr: $\Theta^2 \rightarrow \Theta^2 \ni$

$$\begin{aligned} & \left[(\theta, \theta', e, e') \in \Theta^4, \theta \neq \theta', \right. \\ & r = \mathbf{rdmember}(R_0), \mathbf{len}(M_{\text{type}}(r)) > 2, \\ & i = \mathbf{rdint}(0, \mathbf{len}(M_{\text{type}}(r)) - 3), \\ & j = \mathbf{rdint}(i + 1, \mathbf{len}(M_{\text{type}}(r)) - 2), \\ & Q = \left\{ x: x \in \theta, \right. \\ & \quad \left. (n, k, r = (r_1, \dots, r_{17}), \mu) \in \mathbb{N}^2 \times R_0 \times M, \right. \\ & \quad \left. i \leq n \leq j, \mu = M_{\text{type}}(r)[n], \right. \\ & \quad \left. x \in C_0^{\text{PRM}}(k, r, \mu) \right\} \ni \\ & \left[(k, r, \mu) \in \mathbb{N} \times R_0 \times M, x \in C_0^{\text{PRM}}(k, r, \mu), \right. \\ & \quad \left. y \in C_0^{\text{PRM}}(k, r, \mu), x \in Q, y \in Q \right] \Rightarrow x = y, \\ & B = \left\{ x: x \in \theta', \right. \\ & \quad \left. (n, k, r = (r_1, \dots, r_{17}), \mu) \in \mathbb{N}^2 \times R_0 \times M, \right. \\ & \quad \left. i \leq n \leq j, \mu = M_{\text{type}}(r)[n], \right. \\ & \quad \left. x \in C_0^{\text{PRM}}(k, r, \mu) \right\} \ni \\ & \left[(k, r, \mu) \in \mathbb{N} \times R_0 \times M, x \in C_0^{\text{PRM}}(k, r, \mu), \right. \\ & \quad \left. y \in C_0^{\text{PRM}}(k, r, \mu), x \in B, y \in B \right] \Rightarrow x = y, \\ & e = (\theta \setminus Q) \cup B, \\ & e' = (\theta' \setminus B) \cup Q \Big] \Rightarrow \\ & \mathbf{swapmidpeionr}(\theta, \theta') = (e, e') \end{aligned}$$

Given a schedule pair (θ, θ') , **swapmidpeionr** returns a schedule pair (e, e') identical to (θ, θ') except that for a randomly selected requirement r and two randomly selected mission event instances μ and μ^* of type r_4 all prototype-event instantiations for r inclusively between μ and μ^* belonging to θ are swapped with all prototype-event instantiations for r inclusively between μ and μ^* belonging to θ' . (See Revisions and Changes Digest item 19, page 67.)

Definition 117:

$$\begin{aligned} & \mathbf{randsvcs}: \mathbb{N}^2 \times R_0 \times M \rightarrow \wp(\mathbb{N} \times A_0 \times \mathbb{N}^2) \ni \\ & \forall (n, k, r = (r_1, \dots, r_{17}), \mu = (\mu_1, \dots, \mu_5)) \in \\ & \quad \mathbb{N}^2 \times R_0 \times M, \\ & (t_p, a, s, d) \in \mathbf{randsvcs}(n, k, r, \mu) \Rightarrow \end{aligned}$$

$$\begin{aligned}
 &\mu_1 = r_2, \mu_2 = r_4, \\
 &k < \mathbf{len}(r_3), n < \mathbf{len}(r_3[k]), \\
 &t_p = \mathbf{tref}(r, \mu), \\
 &(\bullet, s_1, s_2, d_1, d_2) = r_3[k][n], \\
 &Q = \left\{ (t_p, a, s^*, d^*) \in \mathbb{N} \times A_0 \times \mathbb{Z} \times \mathbb{N} : \right. \\
 &\quad \exists \mu^V \in M \ni \\
 &\quad \left. (t_p, a, s^*, d^*) \in Y_{\max}^I(r, k, n, \mu, \mu^V) \right\}, \\
 &(t_p, a, s', d') = \mathbf{rndmember}(Q), \\
 &\text{if } \zeta = [s', (s' + d')] \cap [s_1, (s_2 + d_2)], \text{ then} \\
 &s = \mathbf{rndint}(\zeta^-, (\zeta^+ - d_1)), \text{ and} \\
 &\text{if } d_{\max} = \mathbf{min}(\{d_2, (\zeta^+ - s)\}), \text{ then} \\
 &d = \mathbf{rndint}(d_1, d_{\max})
 \end{aligned}$$

Given (n, k, r, μ) , $\mathbf{rndsvcs}(n, k, r, \mu)$ is a set of randomly selected service instantiations for service $r_3[k][n]$ relative to mission event instantiation μ . (See Revisions and Changes Digest item 20, page 67.)

Definition 118:

$$\begin{aligned}
 &\mathbf{rndpeis}: \mathbb{N} \times R_0 \times M \rightarrow \wp(\mathbf{codomain}(Y^I)) \ni \\
 &\forall (k, r = (r_1, \dots, r_{17}), \mu = (\mu_1, \dots, \mu_5)) \in \\
 &\quad \mathbb{N} \times R_0 \times M, \\
 &\xi \in \mathbf{rndpeis}(r, k, \mu) \Rightarrow \\
 &\mu_1 = r_2, \mu_2 = r_4, k < \mathbf{len}(r_3), \\
 &\xi \text{ is a sequence having } \mathbf{len}(r_3[k]) \text{ elements, and} \\
 &\left[[n \in \mathbb{N}, n < \mathbf{len}(\xi)] \Rightarrow \xi[n] \in \mathbf{rndsvcs}(n, k, r, \mu) \right]
 \end{aligned}$$

Given (k, r, μ) , $\mathbf{rndpeis}(k, r, \mu)$ is a set of randomly selected prototype-event instantiations for prototype event $r_3[k]$ relative to mission event instantiation μ .

Definition 119:

$$\begin{aligned}
 &\Theta_{\mathbf{RND}}: \mathbb{N}^+ \rightarrow \wp(\Theta) \ni \\
 &\forall n \in \mathbb{N}^+, \exists Q \subseteq \Theta \ni |Q| = n \text{ and} \\
 &\theta \in Q \Rightarrow \forall p \in \theta, \\
 &\quad \exists (k, r = (r_1, \dots, r_{17}), \mu = (\mu_1, \dots, \mu_5)) \in \\
 &\quad \quad \mathbb{N} \times R_0 \times M \ni \\
 &\quad \mu_1 = r_2, \mu_2 = r_4, \\
 &\quad k = \mathbf{rndint}(0, \mathbf{len}(r_3) - 1), \text{ and} \\
 &\quad \exists i \in \mathbb{N} \ni \\
 &\quad \quad i < \mathbf{len}(M_{\mathbf{type}}(r)) \ni \\
 &\quad \quad \mu = M_{\mathbf{type}}(r)[M_{\mathbf{skips}}(r, 0)[i]], \text{ and} \\
 &\quad \quad p \in \mathbf{rndpeis}(k, r, \mu), \\
 &\text{and } \Theta_{\mathbf{RND}}(n) = Q
 \end{aligned}$$

$\Theta_{\mathbf{RND}}(n)$ returns a set of n randomly generated schedules.

5. Optimal Schedule-Generation Algorithm

The definitions given in Section 4 permit a precise specification of an algorithm for generating optimal solutions for

the NASA space-data communications scheduling problem. These definitions encompass functions for generating random permissible solutions, creating mutations of members of the working population, creating children of pairs of members of the working population using the “genetic crossover” mechanism, and evaluating the fitness of members of the working population. There are many, a very great many, different allowable variants on these functions and therefore a very great many different variants on the algorithm to be specified below. These functions could be replaced or augmented with other allowable functions that reflect more sophisticated genetic mutation and crossover mechanisms including, in particular, additional safeguards against possible premature convergence as discussed in the literature on genetic algorithms. Such refinements are potentially limitless and are beyond the scope of this disclosure. (See Revisions and Changes Digest item 21, page 67.)

5.1. Specification of Optimal Schedule-Generation Algorithm

Algorithm 1 (Optimal-Schedule Generation Algorithm):

1. Assume given:
 - (a) $\nu \in \mathbb{N}^+$ is the run time limit in units of seconds.
 - (b) $n_0 \in \mathbb{N}^+$ is the nominal working size of the population at the beginning of each iteration of the algorithm.
 - (c) $\Pi = \Theta_{\mathbf{RND}}(n_0)$, the initial, randomly selected population of schedules.
 - (d) $\psi \in \mathbb{N}^+$, the number of steps in which new members of the population are generated in each iteration of the algorithm, i.e., the number of steps starting with step 4 and ending with step 15.
 - (e) $\alpha \in \mathbb{N}^\psi$, a tuple having ψ elements \ni
 - i. $\forall j \in \{1, \dots, \psi\}, \alpha_j \in \mathbb{N}$ is the number of new candidate members to be added to the schedule population in step $j + 4$ in the algorithm.
 - ii. $\sum_{j \in \{1, \dots, \psi\}} \alpha_j = n_0$.
- (f) $0 \leq \tau \in \mathbb{R}$, a small value to represent a policy or judgment as to how close to perfect a schedule must be to be considered “good enough” to exit the algorithm. τ normally would be set small enough to ensure that the algorithm always ran for the maximum allowed run time ν .

2. Let $\Pi' = \emptyset$. In each iteration of the algorithm, Π' will accumulate members to be added to the present population, from which combination the n_0 best schedules will be extracted to compose the next generation.
3. $\forall \pi \in \Pi$, let $\pi' = \mathbf{cutexcesspei}(\pi)$ and let $\Pi = (\Pi \setminus \{\pi\}) \cup \{\pi'\}$.
4. $\forall j \in \{1, \dots, \psi\}$, randomly form $\Pi_j \subset \Pi \ni |\Pi_j| = \alpha_j$.
5. $\forall \pi \in \Pi_1$, let $\pi' = \mathbf{slipsvc}(\pi)$, and let $\Pi' = \Pi' \cup \{\pi'\}$.
slipsvc (Definition 106) provides a “mutation” mechanism, where parts of an “organism’s” “genome” are modified to produce an offspring, which is then incorporated into Π' .
6. $\forall \pi \in \Pi_2$, let $\pi' = \mathbf{chngsvcdur}(\pi)$, and let $\Pi' = \Pi' \cup \{\pi'\}$.
chngsvcdur (Definition 107) provides a “mutation” mechanism, where parts of an “organism’s” “genome” are modified to produce an offspring, which is then incorporated into Π' .
7. $\forall \pi \in \Pi_3$, let $\pi' = \mathbf{chngsvستا}(\pi)$, and let $\Pi' = \Pi' \cup \{\pi'\}$.
chngsvستا (Definition 108) provides a “mutation” mechanism, where parts of an “organism’s” “genome” are modified to produce an offspring, which is then incorporated into Π' .
8. $\forall \pi \in \Pi_4$, let $\pi' = \mathbf{chngsvcant}(\pi)$, and let $\Pi' = \Pi' \cup \{\pi'\}$.
chngsvcant (Definition 109) provides a “mutation” mechanism, where parts of an “organism’s” “genome” are modified to produce an offspring, which is then incorporated into Π' .
9. $\forall \pi \in \Pi_5$, let $\pi' = \mathbf{replacepei}(\pi)$, and let $\Pi' = \Pi' \cup \{\pi'\}$.
replacepei (Definition 110) provides a “mutation” mechanism, where parts of an “organism’s” “genome” are modified to produce an offspring, which is then incorporated into Π' .
10. Let $Q = \mathbf{RND}(\frac{1}{2}\alpha_6, \Pi_6^2) \ni (\pi, \theta) \in Q \Rightarrow \neg(\theta, \pi) \in Q$. $\forall (\pi, \theta) \in Q$, let $(\pi', \theta') = \mathbf{swappei}(\pi, \theta)$ and let $\Pi' = \Pi' \cup \{\pi', \theta'\}$.
swappei (Definition 112) provides a “crossover” mechanism, where the “parents” (π, θ) produce “offspring” (π', θ') , parts of whose “genome” are from different parents. Since two new solutions are added for each member of Q , a total of α_6 new solutions will be added. Similarly for each of the subsequent crossover steps below.
11. Let $Q = \mathbf{RND}(\frac{1}{2}\alpha_7, \Pi_7^2) \ni (\pi, \theta) \in Q \Rightarrow \neg(\theta, \pi) \in Q$. $\forall (\pi, \theta) \in Q$, let $(\pi', \theta') = \mathbf{swappeionr}(\pi, \theta)$, and let $\Pi' = \Pi' \cup \{\pi', \theta'\}$.
swappeionr (Definition 113) provides a “crossover” mechanism, where the “parents” (π, θ) produce “offspring” (π', θ') , parts of whose “genome” are from different parents.
12. Let $Q = \mathbf{RND}(\frac{1}{2}\alpha_8, \Pi_8^2) \ni (\pi, \theta) \in Q \Rightarrow \neg(\theta, \pi) \in Q$. $\forall (\pi, \theta) \in Q$, let $(\pi', \theta') = \mathbf{swappeionu}(\pi, \theta)$, and let $\Pi' = \Pi' \cup \{\pi', \theta'\}$.
swappeionu (Definition 114) provides a “crossover” mechanism, where the “parents” (π, θ) produce “offspring” (π', θ') , parts of whose “genome” are from different parents.
13. Let $Q = \mathbf{RND}(\frac{1}{2}\alpha_9, \Pi_9^2) \ni (\pi, \theta) \in Q \Rightarrow \neg(\theta, \pi) \in Q$. $\forall (\pi, \theta) \in Q$, let $\pi', \theta' = \mathbf{swapearlypeionr}(\pi, \theta)$, and let $\Pi' = \Pi' \cup \{\pi', \theta'\}$.
swapearlypeionr (Definition 115) provides a “crossover” mechanism, where the “parents” (π, θ) produce “offspring” (π', θ') , parts of whose “genome” are from different parents.
14. Let $Q = \mathbf{RND}(\frac{1}{2}\alpha_{10}, \Pi_{10}^2) \ni (\pi, \theta) \in Q \Rightarrow \neg(\theta, \pi) \in Q$. $\forall (\pi, \theta) \in Q$, let $(\pi', \theta') = \mathbf{swapmidpeionr}(\pi, \theta)$, and let $\Pi' = \Pi' \cup \{\pi', \theta'\}$.
swapmidpeionr (Definition 116) provides a “crossover” mechanism, where the “parents” (π, θ) produce “offspring” (π', θ') , parts of whose “genome” are from different parents.
15. Let $\Pi'' = \Theta_{\mathbf{RND}}(\alpha_{11})$. Let $\Pi' = \Pi' \cup \Pi''$.
This adds to the population at most α_{11} new members randomly selected from Θ .
16. Find $\Pi^\dagger \subseteq \Pi \cup \Pi' \ni |\Pi^\dagger| = n_0$ and $[\pi_1 \in \Pi^\dagger, \pi_2 \in (\Pi \cup \Pi') \setminus \Pi^\dagger] \Rightarrow \mathbf{fitness}(\pi_1) \leq \mathbf{fitness}(\pi_2)$. Set $\Pi = \Pi^\dagger$ and set $\Pi' = \emptyset$.
17. Find $\pi_1 \in \Pi \ni \zeta \in \Pi \Rightarrow \mathbf{fitness}(\zeta) \geq \mathbf{fitness}(\pi_1)$. π_1 is the best member of Π . If $\mathbf{fitness}(\pi_1) < 1 + \tau$ or run time exceeds the limit ν , go to step 18; otherwise, go to step 4.
18. Output the best schedule π in Π and exit.

5.2. Schedule-Generation Algorithm: Internal Parameters

Every successive generation of the population of schedules retains the best members of the previous generation combined with the new members added in the course of running the algorithm. The best member of a generation will be at least as fit as any member of the preceding generation, and

consequently the fitness of the best member of each generation will be a monotonic function of processing time (or the iteration count) (see Appendix A (page 44)).

The algorithm as specified in the previous subsection does not stipulate the values of the internal parameters (represented by the tuple α), and is silent on exactly how they should be chosen. There is no obvious relationship between the values selected and the performance that should be expected of an implementation of the algorithm, although a method of finding a performance-optimizing set of choices for those values would be, potentially, highly advantageous.

While any reasonable choices of the values of the above internal parameters would not prevent an implementation of the algorithm from reaching an optimal schedule for a given scheduling scenario, other choices might improve performance. In theory, while keeping constant (a) the seeds for the random-number generator, (b) the run time, and (c) the computing resources between runs, runs of the algorithm using different choices of the values of the internal parameters may not find solutions with the same fitness; that is, some of the choices may be significantly more effective in finding optimal solutions with better fitness scores. It should be noted that these internal parameters (as represented by the tuple α) are not the only internal parameters that might be defined. For example, in the mutation steps 5 through 9, the number of places in the individuals' genome that are modified to produce new individuals could be adjusted to reveal the effect on the algorithm performance.

If, for a given representative scheduling scenario, experimental runs of the implementation using a variety of choices for the internal-parameter values revealed a significant performance advantage for a particular choice, it would be valid, absent any further insight or data, to use that choice when running the implementation for other scheduling scenarios. The idea would be that a random or uninformed choice of the values is not likely to be better than a choice that has been found to be, for some representative scheduling scenario, the best one of a set of tested alternatives.

Section 6 will specify an algorithm by which, for any given scheduling scenario, an optimal choice of the values for the internal parameters may be found, assuming such an optimum exists (where "optimum" is again used in the sense indicated in Section 2.4.7 on page 15). The optimal choice, for any given scheduling scenario, would be one for which the algorithm's performance could not be improved by means of a different choice, and the problem of finding such an optimum will hereinafter be referred to as the $S^\#$ problem.

In Section 7, we will propose an answer to the question of whether there is any reasonable way of relating scheduling scenarios to each other, where a "small" difference between two scheduling scenarios would mean a correspondingly small difference in the optimal choice of the values

for the internal parameters. We will seek to identify a solution space for what we hereinafter call the $S^\#$ problem—the final abstraction of the overall space data-communications scheduling problem—in which, for the implementation of the schedule-generation algorithm specified in Section 5.1, there exists an automated way to preprocess a given scheduling scenario to identify an optimal choice of the internal-parameter values. The remaining question of whether the performance of the overall system will be sensitive to differences in the choice of the values of the internal parameters will be left to future work—likely entailing considerable computational effort rather than theoretical analysis.

6. Internal-Parameter Optimization: The $S^\#$ Problem

We will now take up the problem—which in the previous section was designated the $S^\#$ problem—of finding an optimal choice of the values of the schedule-generation algorithm's internal parameters for a given scheduling scenario, thereby to optimize the schedule-generation algorithm for solving that scheduling scenario.

6.1. The $S^\#$ Problem: Introduction

The internal parameter-optimization algorithm to be specified in Section 6.3 will employ the same probabilistic search concepts presented in Section 5 in specifying the schedule-generation algorithm. As before, a population of solutions of the optimization problem will be evolved iteratively, and on each iteration the fitness of each member of the population will be determined. Not all, but just the fittest members of each generation, will be allowed to survive into the next generation.

By definition, each member of the population is not a schedule (as in the schedule-generation algorithm itself), but rather a choice, e , of values of the internal parameters of the schedule-generation algorithm, and choice e will remain fixed until the schedule-generation application program produces the best possible (optimal) schedule for the given scheduling scenario γ . The fitness of each member of the evolving population of such choices e would be a numerical value representing the performance of the system. By definition, the performance of the system (given the choice e) will be the fitness score of the best schedule that can be produced by the schedule-generation algorithm in a prescribed amount of processing time, with prescribed computing resources.

During the entire iterative process of finding the best solution (i.e., the best choice of the values of the internal parameters of the schedule-generation algorithm), the scheduling scenario will remain fixed, and at the end of the iterative

process, the choice of the internal parameter values that resulted in the best performance is considered to be optimal.

This abstracted search problem—the S^\sharp problem—will also have its own internal parameters, one of which is the prescribed amount of processing time allowed for the above iterative process to produce a solution. Additional internal parameters will be described below. While a subsidiary problem could be defined for the optimization of *these* parameters, it will be seen that this subsidiary problem would also have its *own* internal parameters to be optimized, leading to a sub-subsidiary problem of optimizing *these* internal parameters, and so on, without end—a kind of infinite regression. In the case of the NASA scheduling domain, it seems reasonable to ignore these subsidiary problems of optimizing internal parameters of optimization problems, and instead, just make judicious choices for the values of the internal parameters for the problem at hand (i.e., the S^\sharp problem), in the full expectation that the only disadvantage of doing so is that, to reach a solution that has the same fitness, processing time might be greater than it would have been with optimization. This position is further justifiable on the grounds that a one-time effort solving the S^\sharp problem, as proposed in Section 7, can obviate the need to pursue indefinitely a chain of S^\sharp -problem optimizations using the above iterative process.

6.2. The S^\sharp Problem: Definitions

Definition 120 (Set of All Scheduling Scenarios):

$$\Gamma = \left\{ \gamma: \gamma = \left(L^\sharp \subseteq L_0, O^\sharp \subseteq O, I^\sharp \subseteq I, \right. \right. \\ \left. \left. P^\sharp \subseteq P, M^\sharp \subseteq M, R^\sharp \subseteq R_0 \right) \right\} \ni \\ \gamma = \left(L^\sharp, O^\sharp, I^\sharp, P^\sharp, M^\sharp, R^\sharp \right) \in \Gamma \Rightarrow$$

1. $(r_1, \dots, r_{17}) \in R^\sharp, k \in \mathbb{N}, k < \mathbf{len}(r_3),$
 $i \in \mathbb{N}, i < \mathbf{len}(r_3[k]) \Rightarrow$
 $r_3[k][i] = (\lambda, \bullet, \bullet, \bullet, \bullet) \Rightarrow \lambda \in L^\sharp,$
2. $(\lambda_1, \dots, \lambda_7) \in L^\sharp \Rightarrow \exists (r_1, \dots, r_{17}) \in R^\sharp \ni \lambda_1 = r_2,$
3. $(\mu_1, \dots, \mu_5) \in M^\sharp \Rightarrow \exists (r_1, \dots, r_{17}) \in R^\sharp \ni$
 $\mu_1 = r_2,$
4. $((s, s', \lambda, \lambda'), \bullet) \in I^\sharp \Leftrightarrow$
 $\exists (\mu_1, \dots, \mu_5) \in V(s, \lambda_1),$
 $\exists (\mu'_1, \dots, \mu'_5) \in V(s', \lambda'_1),$
 $\exists (r_1, \dots, r_{17}) \in R^\sharp, \text{ and}$
 $\exists (r'_1, \dots, r'_{17}) \in R^\sharp \ni$
 $\lambda_1 = r_2, \lambda'_1 = r'_2,$
 $\mu_1 = r_2, \mu'_1 = r'_2$
5. $(u, \bullet) \in P^\sharp \Rightarrow \exists (r_1, \dots, r_{17}) \in R^\sharp \ni u = r_2$

Definition 121: $\mathbf{fitness}^\sharp: \Gamma \times \mathbb{N}^\psi \times \mathbb{N}^+ \rightarrow \mathbb{R} \ni$

$$(\gamma, e, t) \in \Gamma \times \mathbb{N}^\psi \times \mathbb{N}^+ \Rightarrow$$

1. t is the run time allowed in units of seconds

2. $\mathbf{fitness}^\sharp(\gamma, e, t) = \mathbf{fitness}(\sigma)$ is the fitness score of the best schedule, σ , produced by the schedule-generation algorithm running on the prescribed computing resources during a run interval of length equal to t seconds, for the scheduling scenario γ and the choice e of the values of the internal parameters.

See Definition 102 (page 32) for the definition of the function $\mathbf{fitness}$. ψ , recall, is the number of internal parameters of the schedule-generation algorithm (see Algorithm 1, steps 1d and 1e (page 35)). $\mathbf{fitness}^\sharp$ is the “fitness function” for the S^\sharp problem, which returns 1 for a perfect choice of the values of the internal parameters of the schedule-generation algorithm and larger values for choices that are not so good.

6.3. Algorithm for Solving the S^\sharp Problem

(See Revisions and Changes Digest item 22 on page 67.)

Algorithm 2 (S^\sharp Algorithm):

1. Assume given:
 - (a) $\gamma \in \Gamma$, a scheduling scenario.
 - (b) $\nu \in \mathbb{N}^+$, representing the allowed run time for the schedule-generation algorithm whenever it is executed in the following steps.
 - (c) $\nu^\sharp \in \mathbb{N}^+$, representing the allowed run time for performing iterations of the following steps in the search for the optimal choice of the values of the internal parameters of the schedule-generation algorithm.
 - (d) $\psi \in \mathbb{N}^+$, the number of steps in which new members of the population are generated in each iteration of the algorithm, i.e., the number of steps starting with step 3 and ending with step 7.
 - (e) $\alpha \in \mathbb{N}^\psi \ni \forall j \in \{1, \dots, \psi\}, \alpha_j$ is the number of new candidate members to be added to the population in step $j + 2$ in the algorithm. Let
$$n_0 = \sum_{j \in \{1, \dots, \psi\}} \alpha_j$$
be the nominal working size of the population on each iteration of the algorithm.
 - (f) $\mathbf{ADDSLIMIT}^\sharp \in \mathbb{N}^+$. This is the limit on the number of new members of the population that can be added to the population in any algorithm step.
 - (g) $\Pi = \mathbf{RND}(n_0, \mathbb{N}^\psi) \ni \theta \in \Pi \Rightarrow$
 $\forall j \in \{1, \dots, \psi\}, \theta[j] \leq \mathbf{ADDSLIMIT}^\sharp.$

2. Let $\Pi' = \emptyset$. In each iteration of the algorithm, Π' will accumulate members to be added to the present population, from which combination the n_0 best members will be extracted to compose the next generation.
3. $\forall j \in \{1, \dots, \psi\}$, randomly form $\Pi_j \subset \Pi \ni |\Pi_j| = \alpha_j$.
4. Let $Q \subseteq \Pi_1 \ni |Q| = \alpha_1$.
 $\forall \pi \in Q$,
 - (a) let $\pi' = \pi$,
 - (b) let $j = \mathbf{rndint}(\{1, \dots, \psi\})$,
 - (c) let $\pi'[j] = \mathbf{rndint}(\{1, \dots, \mathbf{ADDSLIMIT}^\sharp\})$,
 - (d) let $\Pi' = \Pi' \cup \{\pi'\}$.

This is a “mutation” mechanism, where one element of an “organism’s” “genome” is modified to produce an offspring, which is then incorporated into Π' .

5. Let $Q = \mathbf{RND}(\frac{1}{2}\alpha_2, \Pi_2^2) \ni$
 $(\pi, \theta) \in Q \Rightarrow \neg(\theta, \pi) \in Q$.
 $\forall(\pi, \theta) \in Q$,
 - (a) let $n = \mathbf{rndint}(\{1, \dots, \psi\})$,
 - (b) let $\pi' = (\pi[1], \dots, \pi[n], \theta[n+1], \dots, \theta[\psi])$,
 - (c) let $\theta' = (\theta[1], \dots, \theta[n], \pi[n+1], \dots, \pi[\psi])$, and
 - (d) let $\Pi' = \Pi' \cup \{\pi', \theta'\}$.

This is a “crossover” mechanism, where randomly many of the first elements of one “organism’s” “genome” are swapped with the same elements in another, resulting in two new members, which are then incorporated into Π' .

6. Let $Q = \mathbf{RND}(\frac{1}{2}\alpha_3, \Pi_3^2) \ni$
 $(\pi, \theta) \in Q \Rightarrow \neg(\theta, \pi) \in Q$.
 $\forall(\pi, \theta) \in Q$,
 - (a) let $n_1, n_2 = \mathbf{rndint}(\{1, \dots, \psi\}), n_1 \neq n_2$,
 - (b) let $\pi' = (\pi[1], \dots, \pi[n_1],$
 $\theta[n_1+1], \dots, \theta[n_2], \pi[n_2+1], \dots, \pi[\psi])$,
 - (c) let $\theta' = (\theta[1], \dots, \theta[n_1],$
 $\pi[n_1+1], \dots, \pi[n_2], \theta[n_2+1], \dots, \theta[\psi])$,
 - (d) and let $\Pi' = \Pi' \cup \{\pi', \theta'\}$.

This is a “crossover” mechanism, where a random section of one “organism’s” “genome” is swapped with the same elements in another, resulting in two new members, which are then incorporated into Π' .

7. Let $Q \subseteq \mathbb{N}^\psi \ni \theta \in Q \Leftrightarrow$
 $\forall j \in \{1, \dots, \psi\}, \theta[j] \leq \mathbf{ADDSLIMIT}^\sharp$.
Let $\Pi' = \Pi' \cup \mathbf{RND}(\alpha_4, Q)$.

This adds α_4 new members to the population randomly selected from \mathbb{N}^ψ .

8. Find $\Pi^\dagger \subseteq \Pi \cup \Pi' \ni$
 $|\Pi^\dagger| = n_0$ and $[\pi \in \Pi^\dagger, \theta \in (\Pi \cup \Pi') \setminus \Pi^\dagger] \Rightarrow$
 $\mathbf{fitness}^\sharp(\gamma, \pi, \nu) \leq \mathbf{fitness}^\sharp(\gamma, \theta, \nu)$.
Let $\Pi = \Pi^\dagger$ and $\Pi' = \emptyset$.
9. Find $\pi \in \Pi \ni \zeta \in \Pi \Rightarrow$
 $\mathbf{fitness}^\sharp(\gamma, \zeta, \nu) \geq \mathbf{fitness}^\sharp(\gamma, \pi, \nu)$.
 π is the best member of Π .
10. If run-time exceeds ν^\sharp , output the best member π in Π and exit; otherwise, go to step 3.

6.4. The S^\sharp Problem: Discussion

As in the case of the schedule-generation algorithm itself, the population of solutions in the S^\sharp algorithm evolve (through the iterative steps of evolutionary search) with a monotonic improvement of the fitness score of the best member of the population toward some evidently limiting value. After some elapsed processing time, the run must be terminated and if the “knee” of the curve that represents the fitness of the best member of the population at the end of each iteration of the algorithm has been passed (see analysis Section 10.2 (page 44)), then the best solution produced to that point is considered to be the optimal solution of the S^\sharp problem.

The question might arise whether the evolving population of solutions might enter a runaway progression of the magnitude of the values of the internal parameters in the execution of the above algorithm. It is quickly seen that this is not a concern: recall that each of the schedule-generation algorithm’s internal parameters represents the number of new schedules that will be allowed to be added to the population in some given step in each iteration of the algorithm. If a choice, e , of the values of the internal parameters included a very large value, the fitness of the best schedule produced within the schedule-generation algorithm’s run-time limit, ν (a given in the S^\sharp algorithm), would be so bad that e likely would not be a member of the next generation.

While no experimentation has been conducted to test it, the working hypothesis is that a diminishing return would result from unbounded increases in the magnitude of any one of the internal parameters, other factors being constant. According to this hypothesis, the performance achieved by the schedule-generation algorithm could be graphed as a function of the value of an arbitrarily chosen one of the schedule-generation algorithm’s internal parameters, keeping other parameters constant. This graph would have a point to the right of which the performance would worsen monotonically. The left-most such point could be found through applying the approaches described herein, but it could only be regarded as pseudo optimal since it would differ from the optimal solution that would be found when the *other* internal parameters were unconstrained as well. Further analysis based

on a model of the performance of the schedule-generation algorithm will be undertaken in Appendix A (Section 10 (page 44)).

7. A Further Abstraction: The S^\sharp Problem

7.1. The S^\sharp Problem: Introduction

To maximize the practicality of the technology disclosed herein, we now consider the S^\sharp problem (described briefly at the end of Section 5)—i.e., the problem of *estimating* an optimal choice of the values of the schedule-generation algorithm’s internal parameters so that it would not be necessary to perform the whole iterative (and computationally expensive) process of solving the S^\sharp problem for every given new scheduling scenario. The S^\sharp objective is to specify a means of easily estimating the best (i.e., optimal) choice of the schedule-generation algorithm’s internal parameters, using (abstracted) information about the given scheduling scenario itself.

No reason has been identified to suspect that the S^\sharp solution space is so ill-behaved as to render it impossible to find a reasonably accurate means of estimating an optimal choice of the values of the algorithm’s internal parameters for “points” in the solution space that are “between” other points for which the optimal choice has actually been calculated (as a solution of the S^\sharp problem). However, the remainder of this section (which describes an approach for solving the S^\sharp problem) may be regarded as somewhat speculative in the sense that (a) the author has performed only a limited amount of relevant experimentation (as mentioned earlier in Section 2.4.6 (page 15)) and (b) the author’s proposed use of certain function-fitting (regression-analysis) techniques, while plausible, is not accompanied by a thorough supporting analysis. It is assumed that available computing platforms are adequate for solving the S^\sharp problem, and that some regression-analysis technology must suffice.

To enable a regression-analysis approach, we make use of a scheduling-scenario characterization function:

Definition 122 (Scheduling Scenario Characterization):

$$\Lambda: \Gamma \rightarrow \mathbb{N}^8 \ni \gamma = (L^\sharp, O^\sharp, I^\sharp, P^\sharp, M^\sharp, R^\sharp) \in \Gamma \Rightarrow \exists (x_1, \dots, x_8) \in \mathbb{N}^8 \ni \Lambda(\gamma) = (x_1, \dots, x_8) \text{ and}$$

1. $Q = \left\{ r = (r_1, \dots, r_{17}) \in R^\sharp: r_4 = \text{“NIL”} \right\} \Rightarrow x_1 = |Q|$
2. $Q = \left\{ r = (r_1, \dots, r_{17}) \in R^\sharp: r_4 \neq \text{“NIL”} \right\} \Rightarrow x_2 = |Q|$

$$3. Q = \left\{ p: \exists r = (r_1, \dots, r_{17}) \in R^\sharp, p \text{ is an element of the sequence } r_3 \right\} \Rightarrow x_3 = |Q|$$

$$4. Q = \left\{ \mu: \exists r = (r_1, \dots, r_{17}) \in R^\sharp, \mu = (\mu_1, \dots, \mu_5) \in M^\sharp, \mu_2 = r_4 \neq \text{“NIL”} \right\} \Rightarrow x_4 = |Q|$$

$$5. x_5 = |L^\sharp|$$

$$6. x_6 = |O^\sharp|$$

$$7. x_7 = |I^\sharp|$$

$$8. x_8 = |P^\sharp|$$

The function Λ produces an eight-dimensional “point” in \mathbb{N}^8 , and, in relation to the S^\sharp problem, we assume that, for two scheduling scenarios γ, γ' , the ordinary Euclidean distance

$$\left(\sum_{i=1}^8 (a'_i - a_i)^2 \right)^{\frac{1}{2}}$$

between two points

$$a = \Lambda(\gamma) = (a_1, \dots, a_8) \in \mathbb{N}^8$$

$$a' = \Lambda(\gamma') = (a'_1, \dots, a'_8) \in \mathbb{N}^8$$

representing the characterizations of γ and γ' , respectively, corresponds to (is commensurate with) the “distance” between γ and γ' .

7.2. Regression-Analysis Approach

In the following paragraphs relative to solving the S^\sharp problem, we assume the availability of an effective regression-analysis technique such as artificial neural networks, Bayesian networks, or support vector machines.

Regression analysis [18, 27]), a collection of well-studied methods of modeling multi-dimensional data interrelationships, is assumed to be viable as a means to derive a function for rapidly estimating, for an arbitrary scheduling scenario, the optimal choice of the values of the internal parameters of the schedule-generation algorithm.

Regression analysis (or simply “regression”), in the broad sense, is analogous to simple least-squares curve fitting with one independent scalar variable and one dependent scalar variable. Regression aims to fit a hypersurface to the set of known data points in the solution space. The best-fitting hypersurface can be expressed as a function that returns the dependent value given the independent value. In the S^\sharp problem, the independent value would be the scheduling scenario (or, normally, the tuple that characterizes a scheduling scenario (i.e., the value returned by the function Λ (see Definition 122 (page 40))), and the dependent value would be the

estimate of the optimal choice of the values of the internal parameters of the schedule-generation algorithm.

The essential, broad steps in applying a regression-analysis approach to the $S^{\#}$ problem are as follows:

Algorithm 3 (Algorithm for Optimal-Internal-Parameters Estimation):

Given:

- A set $\Gamma' \subseteq \Gamma$ of realistic/actual scheduling scenarios. The results of running an implementation of the present algorithm are highly dependent on the number and distribution of these scenarios. If the accuracy of the estimation function generated by this implementation is not deemed adequate, then Γ' would need to be revised and used in a fresh rerun. (Over the past three decades, a great many actual scheduling-problem scenarios have been constructed and solved by the NASA space-data communications scheduling system. These scenarios would be a rich (and probably the most appropriate and reliable) source of data for building the set Γ' .)

Perform the following steps:

1. For each $\gamma \in \Gamma'$,
 - (a) solve the $S^{\#}$ problem computationally, producing the optimal choice e of the values of the internal parameters of the schedule-generation algorithm,
 - (b) compute the characterization $c = \Lambda(\gamma)$.
2. Retain the set Q of known (calculated) points (c, e) , as obtained in step 1.
3. Randomly assign each member of the set Q to either of two (approximately equally numerous) disjoint sets: a training set Q_{train} and a test set $Q_{\text{test}} \ni$

$$|Q_{\text{train}}| - |Q_{\text{test}}| \in \{-1, 0, 1\}$$

4. Perform regression analysis using the training data set Q_{train} , resulting in the determination of the internal-parameters-estimation function that best fits the members of Q_{train} .
5. Using the test data set Q_{test} , test and verify the estimation function derived in step 4.
6. If the derived function passes the test, let f designate the derived function (which represents the fitted hypersurface) and exit indicating success. If the derived function fails the test, then exit indicating failure, calling upon the user to alter the given set of actual/realistic problem scenarios (e.g., by increasing their number or variety) (noting that this alteration gives an altered problem) and rerun the algorithm.

The derived function f estimates the optimal choice of the internal parameters of the schedule-generation algorithm, given any scheduling scenario. This function can be incorporated into a fielded scheduling system to maximize overall performance, and can be used as specified in Section 7.3.

7.3. Operational Use of Derived Estimation Function

The resulting tested and verified estimation function (specified as in Algorithm 3) would then become a tool for operational use within a fielded data-communications scheduling system. The routine use of this tool would involve the following straightforward steps:

Process 1 (Operational Use of Estimation-Function):

1. Prepare a scheduling scenario γ .
2. Supply the characterization $\Lambda(\gamma)$ as input to the estimation function.
3. Capture the estimation-function output e —the estimate of the optimal choice of the schedule-generation algorithm's internal parameters for scheduling scenario γ .
4. Use e in configuring the schedule-generation algorithm for an execution run to produce an optimal schedule for γ .

7.4. The $S^{\#}$ Problem: Discussion

The regression analysis technology called for in Algorithm 3 is associated with extensive research and application literature [2, 4, 6, 9, 14, 16, 21, 23, 24, 27, 30]. For the overall algorithm optimization approach specified in Section 7 for the $S^{\#}$ problem, it may be unjustifiable to assume the ready applicability of off-the-shelf applications. Effective use of the relevant techniques and available applications may require trial-and-error efforts and/or the guidance of experts.

It is explicitly assumed herein that:

- optimizing the internal parameters of the schedule-generation algorithm (see Section 6) is feasible
- the relationship between the independent variables (the problem-scenario characterization) and the dependent variables (the solution found with fixed computing resources) is smooth enough to support approximation by means of some available regression analysis technique analogous to a standard curve-fitting technique .
- the optimization would be effective in the following sense: a system that implemented the schedule-generation algorithm would require less computing resources and have more rapid response in producing high quality schedules, if it took advantage of the

optimization of the choice of the values of the internal parameters, than if otherwise.

This set of assumptions has been tested by the author only preliminarily (relative to the prototype implementation of the unpublished predecessor of the algorithms in the present disclosure) and may, with experience, prove to be unjustified with respect to the $S^{\#}$ problem. For example, it may be found that, while the first two assumptions are confirmed to be valid, the third one is not: the relationship identified in the second assumption may be found to be essentially flat. It would seem more likely, however, that the relationship identified in the second assumption lacks sufficient smoothness to support any reasonable optimization process using the suggested hypersurface-fitting technologies. A determination on this question would require involvement of experts in any such technology chosen for use.

The effort needed to obtain a usable function for estimating the optimal values of the internal parameters (as described in the present section) would be nontrivial, but it would be a one-time effort with a potentially worthwhile increase in operational efficiency of the scheduling system as a whole. In carrying out the effort, it might be learned that no significant variation in solution quality resulted from different choices for the internal parameters. In that event, the determination could be made that the effort had insufficient return and could be discontinued.

8. Implementation

8.1. Prototype Implementation of Predecessor Algorithm

The author implemented the unpublished predecessor of the foregoing algorithms (see Sections 5 and 6) for a prototype automated interference-mitigation scheduler and tested it with input data for selected scheduling scenarios for three actual NASA missions. For these limited cases, the prototype (some 54,000 lines of C++ code) performed with efficiency at a level adequate for practical use—even when executed on the 1995-vintage Unix workstation available at the time. A limited comparison of results with output from the Network Planning and Analysis System (NPAS) [31] developed at NASA Goddard satisfied the author as to the validity and practicality of the approach.

In implementing the prototype, the author noted the utility of a mathematically precise specification of the algorithms. Such a specification clearly supports implementability. It seems reasonable to believe that an implementation attempted without such a specification but with only a typical set of system requirements (a) would entail considerable risks of software rework as high level requirements became

better understood and fleshed out in detail and (b) would require a multiple of the schedule time and funding that would be sufficient with a specification as precise and complete as the one provided in this disclosure.

8.2. Implementing the Disclosed Methods and Algorithms

The present version of the algorithm improves upon—but maintains the essence of the approach of—its predecessor. In particular, readability, implementability, and solution-space coverage are all improved. The present version should also be more adaptable to accommodate inevitable changes in the communications infrastructure, e.g., changes required to implement possible new capabilities for supporting future exploration of the Moon and Mars.

The implementation of the predecessor algorithm suggests no significant question as to the implementability of the method and algorithm disclosed herein (see Section 5 (page 35)), given that the software-system design process is carried out by individuals with an adequate background in NASA's Space and Ground Networks, mathematics, and appropriate regression analysis techniques (if the implementation of the $S^{\#}$ algorithm (Algorithm 6.3 (page 38)) were planned).

Any programming language that is in use in present software implementation projects in NASA's data-communications network infrastructure, such as Java and C++, possesses characteristics that would assure success in implementing the disclosed algorithms. However, preference should be given to a language that is also supported on an available supercomputer or grid computing system for the purpose of running the scheduling system's internal-parameter optimization method described in Section 6 and especially the further algorithm for deriving an estimator function for the optimal values of the internal parameters as described in Section 7. These algorithms (as distinguished from the schedule-generation algorithm itself) are very compute-intensive and should be carried out on the most powerful available computing system, not on the ordinary computers that would be used for development or operations.

An operational implementation of the herein disclosed schedule-generation algorithm (Algorithm 1 (page 35)) might, in terms of size, be comparable to the prototype implementation of the predecessor algorithm. However, under current NASA system-development guidelines, operational systems must be implemented with more-stringent development standards than were used for the earlier prototype implementation, and, further, must accommodate interfaces with existing operational systems. Consequently, the necessary size of an operational implementation

of a new scheduling system based on the present disclosure is presently undetermined but is likely to greatly exceed that of the prototype.

Many organizations (including NASA) require systems developers to follow a rigid software and systems development methodology, and have adopted one of the recognized “process models” that prescribe methods and practices for software and systems engineering. For example, Capability Maturity Model Integration (CMMI) and ISO 9000 have both been followed by NASA. Nothing in this disclosure is intended to favor or limit or to be incompatible with any choice of process model. Nevertheless, the approach taken in this disclosure is in the category of a “formal method”, where rigorous mathematical language is used in all definitions and in all specifications of algorithms for solving problems in a given problem domain. The author views such a formal (that is, mathematical) approach to be not only inherently advantageous, but also complementary to whatever process model is followed. (See Revisions and Changes Digest item 23 on page 67.)

9. Conclusion

The present disclosure describes an evolutionary (i.e., probabilistic) search strategy as the primary approach for attaining an optimal solution of the scheduling problem in the civilian or military space data-communications network. In terms of computer processing time for a problem domain of this kind (whose solution space is so large that no direct algorithmic prescription or brute-force method will suffice (see solution-space analysis in Subsection 2.2)), a probabilistic search application of the kind specified herein progresses, after an initial rapid improvement in quality, monotonically in an iterative fashion *towards* (but without any expectation of actually *arriving at*) some evident (but nevertheless unspecifiable) limiting result (relative to some prescribed measure of “goodness” of solutions) that could not be improved upon through any amount of processing.

At any point during processing, the amount of additional processing time that would be necessary to achieve an additional improvement over already-found solutions becomes greater and greater as the search proceeds. Even if no prescriptive method exists by which to find an optimum in any absolute sense, a probabilistic search strategy can approach arbitrarily close to the limiting result, given unlimited processing resources and time. But, as was observed in Subsection 2.4.5, it is not known how to determine how close to the optimum is the best solution attained at any intermediate point in the processing. The existence of a limiting result (an optimum) seems intuitive, but in practice and in theory, the limiting result cannot assuredly be attained.

Nor can the limiting result actually be specified directly by any method—otherwise, logically, an optimum solution itself would be at hand by the same method.

Nevertheless, it is well known that probabilistic search techniques and methods of the kind described herein can be used to reach optimal solutions for scheduling problems, which leaves an opening for the herein disclosed attack on the space data-communications scheduling problem. The rigorous specification of a system based on these probabilistic search techniques and methods is presented fully herein for the NASA space data-communications scheduling problem, with no known previous equivalent. While evolutionary search techniques have been proposed (e.g., see [25, 28, 32, 12]), no other true optimizing scheduler for this problem domain is known to have been fully specified.

A number of constraints must be considered in designing any system that solves the space data-communications scheduling problem. The methods and algorithms disclosed herein incorporate, among others, the RF-interference mitigation constraint, with the objective of assuring that the system generates high quality schedules that accomplish overall goals of the space data-communications infrastructure. How and to what degree interference predictions should finally constrain schedules represents an issue in the design of such a system. While the algorithm as disclosed herein does satisfy RF-interference mitigation constraints, it does not explicitly provide for fine-grained control of this factor; however, a modification to do so could be incorporated without difficulty. Fine-grain control of this constraint could, for example, include a further parameter in the definition of *mission event* (see Definition 46 (page 22)) to indicate whether or not to apply the constraint for a prescribed instance of a prototype event relative to that mission-event instance.

While the primary context of the present disclosure relates to NASA, the method and algorithm have broader applicability, and, despite domain differences, should readily be adaptable for the military context.

Two abstractions related to the overall problem of devising the most cost-effective possible system for generating optimal schedules were developed in Section 6 (page 37) (the $S^\#$ problem) and Section 7 (page 40) (the $S^{\#\#}$ problem). In presenting the former abstraction, Section 6 described a method and algorithm that increase the efficiency of the search for the optimal schedule given a scheduling scenario γ , by applying an iterative process for determining the optimal choice of the values of the schedule-generation algorithm’s internal parameters. For the given scheduling scenario γ , any other choice of the values of the schedule-generation algorithm’s internal parameters would mean either decreasing the expected quality of the generated schedule or increasing the expected search time for a schedule of a given quality (relative to some prescribed measure of “goodness”). In presenting the latter ab-

straction (in a somewhat speculative vein unsupported by experimental results), Section 7 builds upon the method and algorithm in Section 6 and describes a further method and approach by which to obtain an estimator function that, given a scheduling scenario (or its characterization via a characterization function), would return an estimate of the optimal choice of the values of the schedule-generation algorithm's internal parameters, thus assuring (in the full embodiment and application of the technology disclosed herein) the most cost-effective possible system for operational use in generating optimal, constraint-satisfying schedules for the civilian or military space data-communications infrastructure.

Appendix A explores an assumed model of the performance of the probabilistic search techniques described in the main body of the present disclosure. Analysis of the model imparts understanding and insight into the issue of how the disclosed evolutionary-search algorithm's internal parameters might be set to maximize performance of the system in operational use.

Finally, Appendix B describes a class of problem domains (the *Type-G problem domains*), which encompasses a very broad range of optimization problems including the space data-communications scheduling problem, among many others. A rigorous specification of algorithms and methods for reaching optimal solutions for problems of Type G is disclosed. The disclosed specification affords to developers an efficient implementation path for developing systems to solve such problems.

10. Appendix A. Algorithm Performance

10.1. Best-Solution Fitness (Function of Algorithm-Iteration Count)

To gain insights into the nature of the optimization attainable by the schedule-generation algorithm (Algorithm 1 (page 35)) and the S^\sharp algorithm (Algorithm 2 (page 38)), we will assume and analyze a model for the performance of the schedule-generation algorithm. We will assume that for a given scheduling scenario γ and a given choice e of the values of the internal parameters of the schedule-generation algorithm, the solution (schedule) fitness plotted against the iteration count during a run of the algorithm would be representable by a function having the form:

$$f(p) = \frac{v}{(p-u)^z + w} + q \quad (1)$$

where the independent variable $p \in \mathbb{N}^+$ represents the iteration count during a run of the algorithm, and the depen-

dent variable $f(p)$ represents the fitness of the best schedule in the schedule population at the end of that iteration. The values of the parameters $v, u, w, q, z \in \mathbb{R}$ determine the precise curve that approximates the performance of the schedule-generation algorithm (for the given scheduling scenario γ and given choice e of the values of the internal parameters). The rationale for choosing a function with the form of Equation 1 is partly empirical, but is herein unexplored except for Subsection 10.2

Figure 4 (page 45) illustrates an instance of the function f (with particular values for the parameters v, u, w, q , and z) along with the derivative of f with respect to p :

$$\frac{df(p)}{dp} = \frac{-vz(p-u)^{z-1}}{((p-u)^z + w)^2} \quad (2)$$

Note that the value of $f(0)$ is finite (assuming $(0-u)^z + w \neq 0$), corresponding to the fact that the population of randomly formed solutions at the initialization of the run of the schedule-generation algorithm (Step 1c, page 35) will have a range of fitness scores, all of which will be finite values. The intersection of the function f with the vertical axis represents the best score in the range of scores of the members of the initial population. Of course, a negative iteration count is meaningless and so the model f for the performance of the algorithm as a function of algorithm-iteration count has no meaning to the left of the vertical axis.

An everywhere differentiable monotonic function (such as f) has a monotonic derivative, and in the case of the present model, the derivative is always negative, corresponding to the fact that the assumed fitness model is a function whose slope is always negative—corresponding, in other words, to the fact that the fitness, in general, improves with increasing iterations of the schedule-generation algorithm. Note that the rate of improvement decreases with increasing iterations of the schedule-generation algorithm.

10.2. Assumed-Model Versus Actual Performance

Whether such a function (Equation 1) could be a fair representation of the actual performance of the schedule-generation algorithm makes a reasonable question that is difficult to answer in the affirmative, but it can be argued that at least some such function would be a *worst-case* representation.

The actual performance of the schedule-generation algorithm, as previously indicated, is, for a given run of the algorithm, a discrete (and monotonic) function of the iteration count during the run. That is, the quality of the best schedule in the population at the end of an iteration will be the ordinate of a discrete point whose abscissa is the iteration count, and,

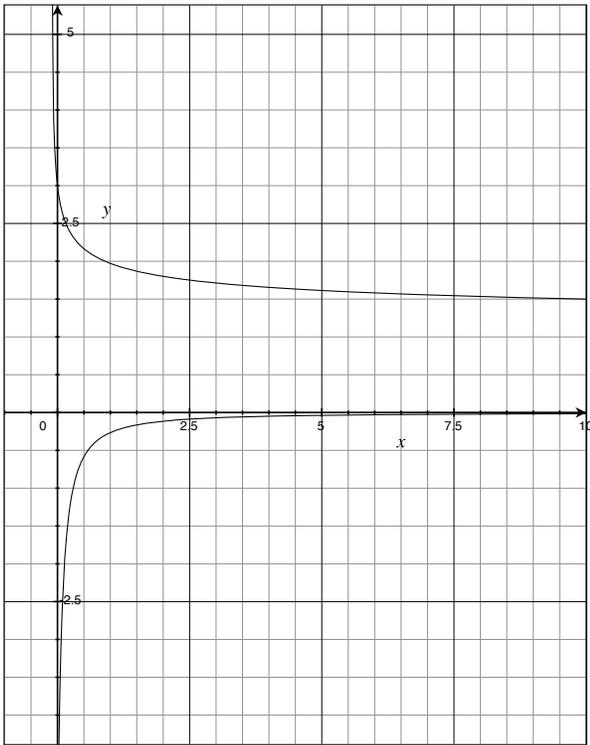


Figure 4. Best-Solution Fitness modeled as a function of the schedule-generation algorithm iteration count (upper curve) (Equation 1), with its derivative (lower curve) (Equation 2). Points on the graph to the left of the vertical axis are to be ignored, since they are meaningless in relation to the actual performance of the algorithm. The *model* for fitness is assumed to be a continuous function, whereas the *actual* performance is a set of discrete points, one for each integer representing the schedule-generation algorithm iteration count.

plotted against iteration count, all such points resulting from the run will be separate dots on the graph, as illustrated in Figure 5. Since the performance of the schedule-generation algorithm has a monotonic relation to the iteration count, if g is the set of points representing the performance of any given run of the algorithm, then there exists a model—i.e., an instance f_{worst} of Equation 1 (with some choice of the values of the parameters $v, u, w, q,$ and z)—such that

1. $\exists x \in \text{dom}(g) \ni g(x) = f_{\text{worst}}(x),$ and
2. $\forall x \in \text{dom}(g) \ni g(x) \leq f_{\text{worst}}(x)$

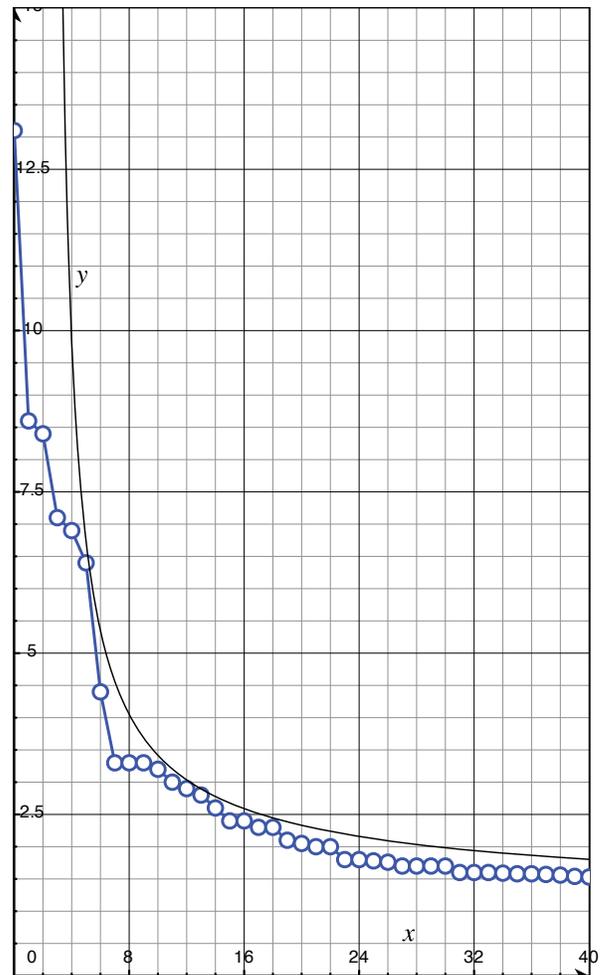


Figure 5. Best-Solution Fitness versus iteration count. The discrete points in the lower graph represent a hypothetical run of the schedule-generation algorithm. The upper curve is a graph of Equation 1, the assumed model of fitness as a function of iteration count during a run of the algorithm, with choices for the values of the parameters $v, u, w, q,$ and z so as to obtain a best-fitting curve of the form of Equation 1 having the discrete points as a lower bound.

and we can refer to this instance of Equation 1 (see Figure 5) as the *worst-case model* of the actual performance. In the remaining subsections of this appendix, the worst-case model can be considered to be the subject of the discussion.

Two further observations are offered regarding Figure 5. First, the discrete points in the lower graph are merely representative and are not actual data from a run of the algorithm. However, the lower graph is notionally consistent with actual execution results of genetic-algorithm applications generally (for example, see [28]). Second, the Equation 1 model, even if it is the worst-case model as depicted in the upper graph in Figure 5, suggests a means of judging a trade-off between the quality of the results from running the algorithm and the power of the computing resources (or the processing time) needed. The model represented by the upper graph has a kind of “knee” where the slope of the curve changes more rapidly than for points either to the left or to the right. To the right of the knee, there is a diminishing-returns situation. The farther to the right, the less improvement in schedule quality expected from a run, but the more computing power (or processing time) required to attain that improvement. To the left of the knee, there is a larger gain in improvement for a given increment in additional computing power (or processing time). In view of the diminishing returns of applying more processing resources at the far right end of the graph, one insight gleaned from considering the model (even the worst-case model) is that reaching a judgement concerning a trade-off between computing power (or processing time) and the quality of the results from running the algorithm can be expected to be facilitated by actual experience running the algorithm. Further, such experience would augment understanding as to how quickly the “knee” will be reached, as well as how long it may take to reach the point of negligible expectation of further improvement⁶.

10.3. Fitness as a Function of a Single Parameter

We now consider the behavior of the model when a full run of the algorithm is repeated with a change in the value of one of its internal parameters. In the repeated run, no other change is imposed. For the present discussion, we let the parameter n represent the change in the value of internal parameter α_j (see definition of α in step 1e of the specification

⁶ These expectations are perhaps reminiscent of the *law of diminishing returns*—an interesting relationship between production and effort in the sense studied in economic theory. It may also remind the reader of another diminishing-returns situation found in Einstein’s Special Theory of Relativity in which any constant application of energy applied to increase the speed of, for example, a spacecraft eventually produces a vanishingly small increase in speed as relativistic effects cause the spacecraft mass to increase without bound.

of Algorithm 1 (page 35)). Thus, in each iteration of the algorithm during the repeated run, at step $j + 4$ (noting that step 4 (page 36) is the first step performed in each iteration) the number of candidate schedules to be generated for inclusion in the population will be changed by the value of n .

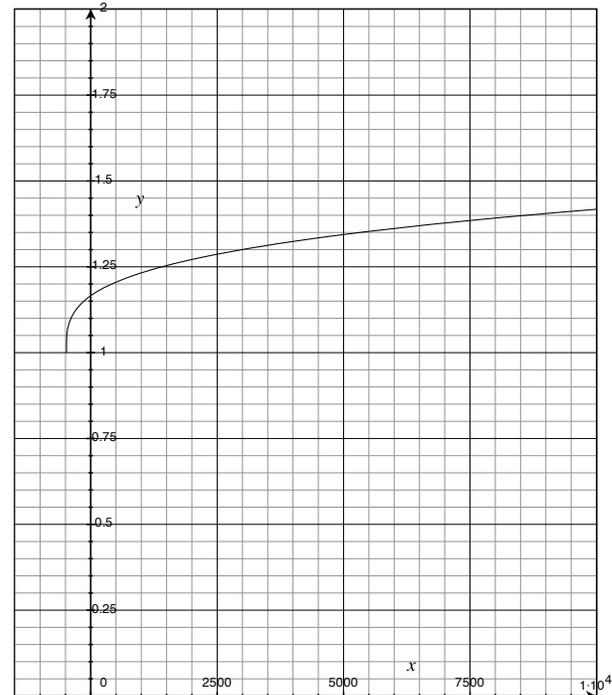


Figure 6. Best-Solution Fitness as a function of the parameter n (representing the change in the fixed number of new members generated at each step) for the assumed model (see Equation 6).

For the initial run (which is assumed to have reached a point in the processing where a large additional amount of processing would not entail a significant expectation of improvement in the fitness of the best schedule, thus reaching an optimum), let p denote the total number of iterations, so far, of the steps of the schedule-generation algorithm, and let m_s denote the number of schedules (i.e., solutions) generated and considered during the run. Then

$$m_s = n_0 + pn_0 \quad (3)$$

where n_0 is as defined in step 1b (page 35).

When the run is repeated, the total run time will be the same (governed by the run-time limit ν defined in step 1a (page 35)). The generation and evaluation of the schedules

created during a given iteration will consume a greater or lesser total amount of computation time, and will decrease or increase the number of iterations of the algorithm during the run, but the total number of schedules that can be generated and evaluated during the repeated run will be the same, equal to m_s , as for the initial run. Thus, with m_s constant between the runs, for the repeated run, we have

$$m_s = n_0 + p(n_0 + n) \quad (4)$$

and so the iteration count can be considered to be a function of $n \in \mathbb{N}$, the independent variable representing the change in the number of new schedules that will be added in some prescribed step of the schedule-generation algorithm:

$$p(n) = \frac{m_s - n_0}{n_0 + n} \quad (5)$$

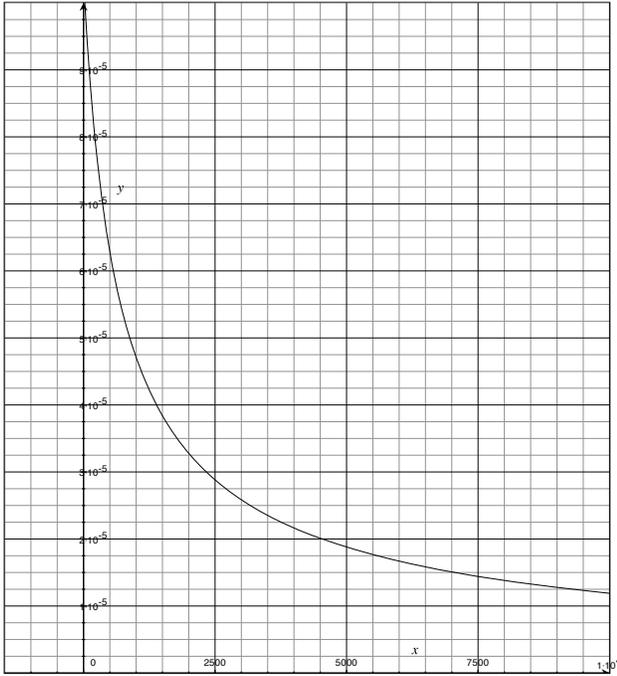


Figure 7. Derivative of best-solution fitness as a function of n (representing the change in the fixed number of new members generated at each step) for the assumed model (see Equation 7).

Equation 1 can now be rewritten to express the fitness function in terms of n :

$$f(n) = \frac{v}{\left(\frac{m_s - n_0}{n_0 + n} - u\right)^z + w} + q \quad (6)$$

illustrated in Figure 6. Notice that the fitness worsens for every positive value of n : the slope of the graph is everywhere positive. The derivative of $f(n)$, given by Equation 7 and illustrated in Figure 7, is always positive, approaching the horizontal axis asymptotically.

$$\frac{df(n)}{dn} = \frac{vz(m_s - n_0)}{\left(\left(\frac{m_s - n_0}{n_0 + n} - u\right)^z + w\right)^2} \frac{\left(\frac{m_s - n_0}{n_0 + n} - u\right)^{z-1}}{(n_0 + n)^2} \quad (7)$$

From these observations it is seen that a larger value of n worsens the fitness by a greater amount than does a smaller one, but the effect diminishes with ever larger values of n .

Thus, when the algorithm's performance is directly related to the number of iterations of the steps of the algorithm, as in the assumed model (Equation 1), the following questions arise:

First, does the model of fitness as a function of n (the change in the number of candidate schedules generated in some prescribed step in each algorithm iteration for inclusion in the population in the repeated run) (see Equation 6 (page 47)) imply that arbitrarily increasing the value of an internal parameter necessarily brings a decrease in the quality of the best solution (schedule)—in comparison to the best schedule that can be produced in the number of iterations performed in the initial run?

Second, does the model imply that *decreasing* the value of an internal parameter in the repeated run necessarily brings an *increase* in the quality of the best solution (schedule)—in comparison to the best schedule that can be produced in the number of iterations performed in the initial run?

In each run of the algorithm (when used as specified in Algorithm 1), values of the internal parameters are held constant (i.e., the number of candidate solutions (schedules) added in each of the algorithm's steps 4 through 15 (page 36) remains the same until the end of the run). At the end of the run (the duration of which will equal the run-time limit ν), the system will output the best schedule in the population. In a repeated run with the increase/decrease of n in the number of candidate schedules generated in each iteration, even if all other parameters are held the same, at any given iteration number, say the k th, the algorithm will necessarily have considered either more schedules or fewer schedules (depending on whether n is positive or negative), and from the first iteration onward, the population of schedules will increasingly diverge from the population in the initial run. Therefore, the model assumed in Equation 1 has limited use in answering the stated questions, but nevertheless affords a useful insight into the setting of the algorithm's internal parameters, as discussed in the next section.

10.4. Fitness as a Function of Elapsed Run Time

It will now be insightful to analyze the behavior of the fitness model (Equation 1) as a function of the elapsed run time of the algorithm, when holding fixed the value of n (i.e., the variable representing the change in the value of a single internal parameter as described in the preceding section). In this analysis, we assume we are given the following:

1. ν , the duration of the algorithm execution run
2. n_0 , the number of schedules added to the population during each iteration of the steps of the algorithm
3. n , the change (relative to a previous run) in the value of a single internal parameter as described in the preceding section
4. m_s , the total number of schedules that can be added and evaluated during a run of duration ν seconds as described in the preceding section

With the above given information, we seek to reformulate Equation 1 to calculate the fitness of the best member of the evolving population of schedules as a function of time.

Since m_s schedules can be created and evaluated by the algorithm during a run of ν seconds duration with the assumed given computing resources, the time required to create and evaluate each schedule (as an overall average) is

$$\frac{\nu}{m_s}$$

When the iteration count is p during a second run of the algorithm, the cumulative number of schedules added will be

$$n_{\text{cum}} = n_0 + (n_0 + n)p$$

It is now possible to calculate how much time has elapsed when the run reaches iteration p :

$$t = n_{\text{cum}} \frac{\nu}{m_s} = (n_0 + (n_0 + n)p) \frac{\nu}{m_s}$$

We can now express the iteration count p as a function of t :

$$p(t) = \frac{m_s t - n_0 \nu}{(n_0 + n)\nu}$$

Thus, Equation 1 can be rewritten to relate fitness to elapsed run time t :

$$f(t) = \frac{v}{\left(\frac{m_s t - n_0 \nu}{(n_0 + n)\nu} - u\right)^z + w} + q \quad (8)$$

illustrated in Figure 8. In the upper curve, the value of n is a positive number, and, in the lower curve, n is a negative number of the same magnitude. The conclusion from these

two curves is that, given any two runs of the algorithm where the first has a larger, and the second has a smaller, number n of schedules added in each iteration, and given any particular elapsed time t during each run, the run that has the smaller value of n will have a better value of the fitness of the best member of the population at time t . The two curves illustrate the fact that the effect is greatest at the beginning of the run and necessarily vanishes at the end when the elapsed time equals ν .

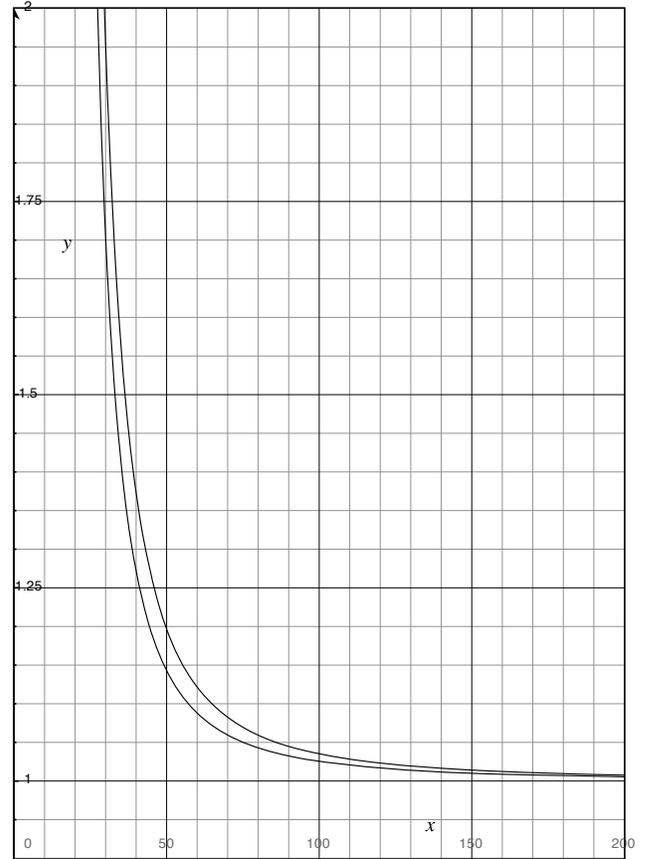


Figure 8. Fitness modeled as a function of the elapsed run time, holding fixed the value n for the change in the value of a single internal parameter. Two runs are illustrated. The upper curve has n set to a positive number, while the lower curve has n set to a negative number of the same magnitude.

However, these relationships are subject to a caveat. In the second run of the algorithm contemplated above, the population starting at the second iteration of the steps of the al-

gorithm begins to diverge from that of the first run, since the number of schedules added in each iteration in the second run will be either greater than or less than the number added in each iteration in the first run. The model described by Equation 1, while arguably representative of the performance of the system during any given run, has no relevance to the differences between two different runs. If Equation 8 or Figure 8 is at all suggestive of some guidance in setting the values of the internal parameters, it would be that smaller rather than “large” values would improve performance.

10.5. Key Insight Afforded by the Assumed Model

Clearly, the model assumed in Equation 1 does imply that increasing or decreasing the number of iterations in a run in a manner that does not result in a difference in the number of candidates generated in any step of the algorithm (by, for example, increasing or decreasing the value of ν) will, in general, correspondingly affect the quality of the solution produced by the algorithm.

Significantly, the model of fitness as a function of the parameter n (see Equation 6 (page 47) and Figure 6 (page 46)) implies that the number of new schedules that will be added in each step of the algorithm is very important: negative values of n result in generating fewer candidate schedules in each iteration, and therefore result in a greater number of iterations during the run. This, together with the analysis in Section 10.4, leads to the hypothesis that experimentation (or, better, the application of the S^\sharp algorithm (Algorithm 2 (page 38))) would show that *the optimal choice of the values of the internal parameters for a given scheduling scenario would entail smaller values rather than larger.*

This is perhaps the most useful insight to be drawn from considering the above model of the performance of the schedule-generation algorithm—namely, that despite the fact that the specification of the algorithm is silent on how to choose the values of the internal parameters, large values would be contraindicated. This insight still does not give quantitative guidance on what constitutes “large” values, however, and therefore does not really substitute for the quantitative guidance that ultimately would be available when a solution to the S^\sharp problem is implemented as described in Section 7 (page 40). But the stated insight suggests that even routine experience running an implementation of the schedule-generation algorithm would eventually lead to a level of practical understanding of at least how *not* to set the values of the internal parameters.

11. Appendix B. The Generalized Algorithms and Methods

11.1. The General Problem

11.1.1. Introduction

In this appendix, we present a generalization of the methods and algorithms (and of the problem domain itself) that were described in Section 5 (page 35), Section 6 (page 37), and Section 7 (page 40). Reaching toward such a generalization is motivated by the recognition (a) that there are many different problem domains where an evolutionary search strategy can be used profitably, and (b) that application developers may benefit from having such a generalization when translating to another domain the methods and algorithms that were described herein for the NASA space-data-communications scheduling problem.

The generalization presented in this appendix is intended to facilitate development of applications for any problem domain matching the essential characteristics of the Type-G problem domain defined below. While the Type-G problem domain is not all-encompassing, it is general enough to be broadly useful and has certain characteristics that facilitate implementation as well as efficient computation. (For example, integer values suffice for the vast majority of all of the calculations that are required for execution of the algorithms.)

This appendix also delineates steps for implementing the generalized methods and algorithms.

Material to be presented below starts with a description of Type-G problems and the approaches for reaching optimal solutions to such problems, and progresses then to descriptions of Type-G problem abstractions that, along with methods to solve them, are designed to support the optimization of those approaches themselves, for use in fielded systems that solve Type-G problems. The abstractions, based on the idea of *Type-G meta problems*, are referred to as the G^\sharp and $G^{\sharp\sharp}$ problems.

11.1.2. Basic Definitions

In the case of the earlier treatment of the NASA space-data communications-scheduling problem, a problem scenario was a set of data structures that, in part, expressed user requirements for data-communications events by which science data (among other kinds of data) could be returned to earth via antennas in the NASA data-communications support infrastructure, or by which commands or other types of data could be received from earth by the user spacecraft, again via antennas in the NASA data-communications support infrastructure. A problem scenario also contained addi-

tional data structures comprising infrastructure constraints, characteristics of support antennas, etc.

Similarly, in the case of the generalized problem domains that we are discussing in the present section, a problem scenario is a finite data structure that expresses requirements and constraints that a problem solution must satisfy.

Definition 123: γ is a *problem scenario of Type G* $\Rightarrow \exists k \in \mathbb{N}^+ \ni \gamma$ is a sequence of exactly k finite data structures that expresses requirements/constraints that are internally consistent and that every allowable solution for γ must satisfy.

Definition 124:

$$\Gamma = \{\gamma: \gamma \text{ is a problem scenario of Type G}\}.$$

Γ is the set of all possible problem scenarios of Type G.

Definition 125: $\Gamma_{\text{dim}}: \mathbb{N}^+ \rightarrow \wp(\Gamma) \ni$

$$\forall(k, \gamma) \in \mathbb{N}^+ \times \Gamma, \gamma \in \Gamma_{\text{dim}}(k) \Leftrightarrow \text{len}(\gamma) = k.$$

$\Gamma_{\text{dim}}(k)$ is the set of all problem scenarios of length k (i.e., of *dimension* k).

Relative to the optimization issues addressed in this disclosure, we will refer to problem domains of Type G defined as follows.

Definition 126: $\Delta = \left\{ \delta: \exists k \in \mathbb{N}^+ \ni \delta \subseteq \Gamma_{\text{dim}}(k) \right\}$.

Definition 127:

δ is said to be a *problem domain of Type G* if and only if $\delta \in \Delta$.

A problem domain of Type G is a set of Type-G problem scenarios all having the same dimension.

Note that the term ‘‘Type-G problem’’, as distinct from the term ‘‘Type-G problem domain’’, will have a more specific definition (see Definition 136 (page 53)).

Definition 128: $\Delta_{\text{dim}}: \Delta \rightarrow \mathbb{N}^+ \ni \forall(\delta, k) \in \Delta \times \mathbb{N}^+$,

$$\Delta_{\text{dim}}(\delta) = k \Leftrightarrow [\forall \gamma \in \delta, \text{len}(\gamma) = k]$$

$\Delta_{\text{dim}}(\delta)$ is the *dimension* of δ , i.e., $\Delta_{\text{dim}}(\delta)$ is a positive integer representing the length of every problem scenario in δ .

We now identify the set of all permissible solutions for Type-G problem domains.

Definition 129 (Set of All Permissible Solutions):

$$\Theta = \left\{ \theta: \exists \delta \in \Delta, \exists \gamma \in \delta \ni \right.$$

θ is a permissible solution for problem scenario γ $\left. \right\}$.

Θ is the set of all permissible solutions for all problem scenarios $\gamma \in \delta \in \Delta$, with no omissions or exceptions. The problem-domain dependent notion of ‘‘permissible’’ is left undefined.

Definition 130 (Set of All Permissible Solutions for a Given Problem Scenario):

$$\Theta_{\text{scenario}}: \Delta \times \Gamma \rightarrow \wp(\Theta) \ni$$

$$\forall(\delta, \gamma) \in \Delta \times \Gamma \ni \gamma \in \delta, \theta \in \Theta_{\text{scenario}}(\delta, \gamma) \Leftrightarrow$$

θ is a permissible solution for problem scenario γ .

Definition 131: $D_0 = \{x: \exists n \in \mathbb{N}^+ \ni x \in \mathbb{Z}^n\}$

D_0 is the set of all tuples whose elements are integers. (See Revisions and Changes Digest item 24 on page 67.)

11.1.3. Associating Generalized Problem Domains with Real Problem Domains

It will be taken for granted herein that the context for discussion relates to a certain class of real-world problems for each of which the following hold:

- it can be stated in some proper manner and given an appropriate working association with some member of Δ ,
- it has a definable solution that can be expressed as a finite data structure, and
- its implementation following the generalized methods and algorithms disclosed herein would flow from the actual problem statement and its associated member of Δ .

The way in which a problem domain in this class might be given an association with a member of Δ , and the specification of the implementation of the actual problem statement in the context of the generalized methods and algorithms, are each beyond the scope of this disclosure.

The foregoing definitions lead to the following observations concerning a problem domain of Type G:

1. There exists a fitness function that assigns to each permissible solution for any given problem scenario a quantitative ‘‘goodness’’ or ‘‘fitness’’ score.
2. Every problem scenario has an optimal solution with reference to a given fitness function.

Type-G problem domains are numerous and varied, and include not only the schedule-optimization problem addressed earlier by means of evolutionary search (genetic algorithms) (Section 5 (page 35)), but also the meta problem designated as the $S^\#$ problem that likewise was addressed by means of evolutionary search (genetic algorithms) (Section 6 (page 37)).

11.1.4. A Note on Applicability

While in theory the disclosed generalized methods and algorithms could be used to reach a solution to many problems for which standard numerical or closed-form methods exist, such usage would be inefficient and would produce less accurate solutions. Nothing in this disclosure is to be construed

so as to obviate common sense in the use of the methods described herein. Only *appropriate* applications are to be contemplated.

There are numerous appropriate applications of the methods and algorithms specified in this appendix for reaching optimal solutions of problem scenarios of Type G. The design-optimization applications in the realm of space missions mentioned in Section 1.6 (page 9) are representative of a wide variety of design-optimization problems, from architecture to automobiles to industrial plants to rail systems to product packaging—to mention only a few that

- can be found in the literature concerning applications of evolutionary search,
- can be classified as problems of Type G, and
- therefore can be solved by a system following the methods and algorithms specified herein.

Scheduling and planning generally can be cast as problem domains of Type G. But the range of Type-G problems is greater yet, and is not likely soon to be fully traversed.

11.1.5. The Essential Evolutionary Search Functions

Genetic algorithms, and evolutionary programming, are applicable to a broad range of problems (see discussion in Section 2.3 (page 13) and Section 2.4 (page 13)), and form the central technical approach for solving Type-G problems as presented herein. The essential mechanisms of evolutionary search (beyond the basic notion of evolving a population of candidate solutions through an iterative process) are—

- fitness
- random selection
- genetic mutation
- genetic crossover

These mechanisms will be embodied in functions defined below.

11.1.5.1. Fitness Functions A crucial part of an evolutionary search algorithm as described herein is the fitness function that will be used to evaluate candidate solutions in the solution space. A valid fitness function for a given problem scenario γ maps each member of the set of all permissible solutions for γ to a real number greater than or equal to 1, where unity is the fitness of a perfect solution. While other choices for the definition of “perfect” fitness are worth considering, the chosen value, 1, affords certain desirable numerical advantages for constructing an actual fitness function. For example, the fitness function defined in Definition 102 (page 32) for the space-data communications

scheduling problem had as constituents a number of independent sub-functions, each defined with fitness in the semi-closed real-number interval $[1, \infty)$, where the final fitness was computed as the product of the values returned by the constituent functions, thus ensuring that the final fitness value always remained in $[1, \infty)$ and, further, that the final fitness value strongly reflected the degrading effect of any one of the constituent values that exceeded 1.

More exactly,

Definition 132 (Set of All Fitness Functions):

$$F_{\text{fitness}}: \Delta \times \Gamma \rightarrow \wp([1, \infty)^\Theta) \ni \\ \forall(\delta, \gamma) \in \Delta \times \Gamma, f \in F_{\text{fitness}}(\delta, \gamma) \Leftrightarrow$$

1. $\text{dom}(f) = \Theta_{\text{scenario}}(\delta, \gamma)$
2. $\forall \theta, \theta' \in \Theta_{\text{scenario}}(\delta, \gamma), \\ f(\theta) < f(\theta') \Rightarrow \theta \text{ is more fit than } \theta'.$

$F_{\text{fitness}}(\delta, \gamma)$ is the set of all possible fitness functions for problem scenario γ in the target problem domain δ . (See Definition 13 (page 19) for the meaning of the notation of the form X^Y , where each of X and Y is a set. In the present definition, $[1, \infty)^\Theta$ designates the set of all functions that map Θ to the semi-closed interval $[1, \infty)$ on the real-number line.)

11.1.5.2. Random Selection The search strategy also employs, for given problem scenario γ for given target problem domain δ , a means to create a set of new candidate members of the working population by either (a) randomly generating permissible solutions for γ or (b) randomly selecting members of $\Theta_{\text{scenario}}(\delta, \gamma)$.

Definition 133 (Selection of a Random Set of Solutions):

$$F_{\text{randomselection}}: \Delta \times \Gamma \rightarrow \wp\left(\left(\wp(\Theta)\right)^{\mathbb{N}^+}\right) \ni \\ \forall(\delta, \gamma) \in \Delta \times \Gamma, f \in F_{\text{randomselection}}(\delta, \gamma) \Leftrightarrow$$

1. $\text{codomain}(f) \subseteq \Theta_{\text{scenario}}(\delta, \gamma),$
2. $\forall n \in \mathbb{N}^+, f(n) = \mathbf{RND}(n, \Theta_{\text{scenario}}(\delta, \gamma)).$

Note that a function belonging to $F_{\text{randomselection}}$ is a “pseudo function” (see remark below Definition 17 (page 20)).

The question may arise as to whether it would be advantageous to “jump-start” the evolutionary search process by somehow including at least one known solution in the initial working population of solutions. Such a known solution might be obtained by employing some special preprocessing step or some more sophisticated method that could match the results achievable, for example, by a human expert. Such an alternate approach, then, differs from the approach prescribed in the above definition of $F_{\text{randomselection}}$, where only a true random selection of (permissible) solutions composes the initial generation. It might be supposed that the alternate approach would increase the expected cost-effectiveness of

the search process (i.e., a better average quality of the best solution produced for a given fixed run time). Even with extensive experimentation and testing, conclusions on this question would be subject to doubt, for no theoretical framework or guidance is in evidence to support the interpretation, analysis, validation, and integration of testing data. For example, an early question would concern whether the known solution produced in the alternate approach had been reached using the same goodness metrics in the same manner as given in the prescribed approach. If not, then how would the difference in the goodness metrics and the manner in which they were applied be characterized and accounted for in the final result of any comparison test? Another source of doubt would be the very means or agent that produced the solution to be used in the alternate approach: in particular, can two human experts always be counted upon to produce the exact same solution for every given problem scenario, and if not, then what is the support for any particular conclusion concerning the value of the alternate approach? The accurate resolution of doubts regarding the validity and cost effectiveness of the alternate approach is difficult to forecast, and the entire problem is beyond the scope of the present disclosure. Suffice it to say that the effort and argumentation required to resolve the doubts would be considerable if not daunting, and ultimately of questionable value even if focused on a particular problem domain. In summary, as part of an operational system, the alternate approach (even if otherwise it is appropriate as a target of research) is difficult to justify on the basis of current theory or knowledge in the general field of evolutionary search, and a more supportable course would be to adhere to the approach given in this disclosure. (See item 25, page 67, in Revisions and Changes Digest section.)

11.1.5.3. Mutation Functions Evolutionary search algorithms also involve functions that generate new candidate members of the working population of solutions using mutation and crossover concepts (see the discussion of genetic algorithms in Section 2.4 (page 13)). To generate a mutation of a member θ of the working population during a search, a new data structure is created representing the new member of the population, i.e., the mutated version of the given member θ . The new data structure (a finite data structure by definition) will be the same as that of the given member θ , except for some deliberate (yet random), problem-scenario-specific alterations made by the mutation function. Once created, the new member is incorporated into the working population on an equal footing with all other members until the member-evaluation phase of the search algorithm is reached during an iteration through the steps of the algorithm. If by chance the mutation produces a child that is identical with its parent, then it will be ignored. It may be noted that the mechanism for making mutations legitimately may not select purely ran-

dom parts of the data structure representing a member of the population, as such a mutation mechanism would likely produce offspring containing uninterpretable/impossible data, rendering them impermissible as solutions of the given problem scenario. Therefore, it may be taken for granted that the mutation mechanism must not lead to nonsensical or otherwise impermissible offspring.

Definition 134: $F_{\text{mutation}}: \Delta \times \Gamma \rightarrow \wp(\Theta^\Theta) \ni \forall(\delta, \gamma) \in \Delta \times \Gamma, f \in F_{\text{mutation}}(\delta, \gamma) \Leftrightarrow$

1. $\text{dom}(f) = \Theta_{\text{scenario}}(\delta, \gamma)$
2. $\text{codomain}(f) \subseteq \Theta_{\text{scenario}}(\delta, \gamma)$
3. $\theta \in \text{rndmember}(\Theta_{\text{scenario}}(\delta, \gamma)) \Rightarrow f(\theta)$ is a random mutation of θ .

$F_{\text{mutation}}(\delta, \gamma)$ is the set of all possible mutation functions for problem scenario γ in the target problem domain δ . Note that this function is a “pseudo function” (again see remark below Definition 17 (page 20)).

11.1.5.4. Crossover Functions The crossover mechanism is defined similarly. A crossover function accepts two randomly selected members of the current working population and produces, in some random yet problem-scenario-specific manner, two new members whose characteristics and fitness will be determined partly by each of their “parents”—the two given members. Each crossover function operates by identifying two random crossover points. The crossover points define segments of the data structure of each of the parents. Once the segments are determined, the two “children” of the parents will be assembled from their parents’ corresponding segments in such a way that the children differ from each other and from each parent—preserving, in that way, the concept of swapping some, but not all, of the chromosomal material between the parents as a means to produce the children. If by chance the children and their parents do not all differ from each other—a result that is not allowed—then it will be necessary to repeat the step for randomly selecting the parents and the segments to be swapped. It may be noted, again, that the crossover points legitimately may not define purely random parts of the data structure representing a parent, as swapping portions defined in that manner would often (perhaps nearly always) produce offspring containing uninterpretable/impossible data, rendering them impermissible. Therefore, it may be taken for granted that the mechanism for selecting crossover points must not lead to nonsensical or otherwise impermissible offspring.

Definition 135: $F_{\text{crossover}}: \Delta \times \Gamma \rightarrow \wp((\Theta^2)^{\Theta^2}) \ni \forall(\delta, \gamma) \in \Delta \times \Gamma, f \in F_{\text{crossover}}(\delta, \gamma) \Leftrightarrow$

1. $\text{dom}(f) = \Theta_{\text{scenario}}(\delta, \gamma)^2$,

2. $\text{codomain}(f) \subseteq \Theta_{\text{scenario}}(\delta, \gamma)^2$,
3. $(\theta, \theta') \in \text{rdmber}(\Theta_{\text{scenario}}(\delta, \gamma)^2), \theta \neq \theta' \Rightarrow f(\theta, \theta')$ is a pair (θ_c, θ'_c) of children generated as crossovers between parents θ and θ' .

$F_{\text{crossover}}(\delta, \gamma)$ is the set of all possible crossover functions for problem scenario γ in the target problem domain δ . Note that every member of $F_{\text{crossover}}(\delta, \gamma)$ is a ‘‘pseudo function’’.

We now seek to formulate the Type-G problem, generally applicable to any problem domain of Type G.

11.2. The Type-G Problem

A *Type-G problem* is a member of the set G :

Definition 136 (Set of All Type-G Problems):

$$G \subseteq \Delta \times \Gamma \times \mathbb{N}^+ \times [0, \infty) \times D_0 \times (\wp(\Theta))^{\mathbb{N}^+} \times [1, \infty)^{\Theta} \times \Theta^{\Theta} \times (\Theta^2)^{\Theta^2} \times \Theta \ni$$

$$g = (\delta, \gamma \subseteq \delta, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G \Rightarrow$$

1. ν represents, in units of seconds, an adequate run time for the search,
2. τ represents a small real value (not necessarily positive) that reflects the user’s judgment or policy as to how close to perfect a solution for the given problem scenario must be to be considered ‘‘good enough’’,
3. α is a vector (i.e., a tuple $(\alpha_1, \alpha_2, \dots)$) consisting of values of the internal parameters of g , with $\text{len}(\alpha) = \Delta_{\text{dim}}(\delta)$,
4. $f_r \in F_{\text{randomselection}}(\delta, \gamma)$,
5. $f_{\text{test}} \in F_{\text{fitness}}(\delta, \gamma)$,
6. $f_m \in F_{\text{mutation}}(\delta, \gamma)$,
7. $f_c \in F_{\text{crossover}}(\delta, \gamma)$, and
8. $\pi \in \Theta_{\text{scenario}}(\delta, \gamma)$.

g is said to be a Type-G problem if and only if $g \in G$. (See Revisions and Changes Digest item 26, page 67.)

It will be useful to identify the Type-G problems for a given Type-G problem scenario γ , as follows:

Definition 137 (Set of All Type-G Problems for a Given Problem Scenario):

$$G_{\text{scenario}} : \Delta \times \Gamma \rightarrow \wp(G) \ni \forall (\delta, \gamma) \in \Delta \times \Gamma,$$

$$G_{\text{scenario}}(\delta, \gamma) = \left\{ (\delta, \gamma, \bullet, \bullet, \bullet, \bullet, \bullet, \bullet, \bullet, \bullet) \in G \right\}.$$

11.3. Type-G Problems: The G-Algorithm

In moving on toward defining an algorithm for solving a Type-G problem, it will be helpful to define a function that embodies certain steps that must be performed by the algorithm relative to a given set of candidate solutions of a given Type-G problem:

Definition 138 (Internal Steps of a G-Algorithm):

$$W_{\text{steps}} : \wp(\Theta) \times G \rightarrow \wp(\Theta) \ni$$

$$\forall (\Pi, g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi)) \in \wp(\Theta) \times G$$

if

1. $\Pi \subseteq \Theta_{\text{scenario}}(\delta, \gamma)$,
2. $n_{\text{population}} = \sum_{j \in \{1, \dots, \text{len}(\alpha)\}} \alpha_j$,
3. $|\Pi| \geq n_{\text{population}}$,
4. $\Pi_1 = \mathbf{RND}(\alpha_1, \Pi)$,
5. $\Pi_2 = \mathbf{RND}(\frac{1}{2}\alpha_2, \Pi^2) \ni (\theta, \theta') \in \Pi_2 \Rightarrow \neg(\theta', \theta) \in \Pi_2$,
6. $\Pi_3 = \left(\bigcup_{\theta \in \Pi_1} \{f_m(\theta)\} \right) \cup \left(\bigcup_{(\theta_1, \theta_2) \in \Pi_2} \{f_c(\theta_1, \theta_2)\} \right) \cup f_r(\alpha_3)$, and
7. $\Pi_4 \subset \Pi_3 \ni$
 - (a) $|\Pi_4| = n_{\text{population}}$ and
 - (b) $[\theta_1 \in \Pi_4, \theta_2 \in \Pi_3 \setminus \Pi_4] \Rightarrow f_{\text{test}}(\theta_1) \leq f_{\text{test}}(\theta_2)$,

then $W_{\text{steps}}(\Pi, g) = \Pi_4$.

The G-algorithm for solving a Type-G problem is an evolutionary-search algorithm specified as follows:

Algorithm 4 (G-Algorithm Specification):

Given:

- $g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G$

Perform the following steps:

1. Let $\Pi = f_r \left(\sum_{j \in \{1, \dots, \text{len}(\alpha)\}} \alpha_j \right)$.
2.
 - (a) Let $\Pi' = W_{\text{steps}}(\Pi, g)$.
 - (b) Find $\pi' \in \Pi' \ni \zeta \in \Pi' \Rightarrow f_{\text{test}}(\zeta) \geq f_{\text{test}}(\pi')$.
 - (c) Set $\pi = \pi'$.
 - (d) Set $\Pi = \Pi'$.
 - (e) If $f_t(\pi) < 1 + \tau$ or runtime exceeds ν , then exit, returning the value π ; otherwise, go to step 2a.

When the G-algorithm is executed to solve a Type-G problem $g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G$, the result is that π has been set equal to the optimal solution for the problem scenario $\gamma \in \delta$. This optimal solution is calculated given α , the vector of the values of the Type-G problem’s internal parameters (where a different choice of their values would generally result in a different solution for the problem scenario γ).

11.3.1. Execution of a G-algorithm

Executing the G-algorithm (Algorithm 4) for a given Type-G problem

$$g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G$$

consists of performing the prescribed steps given in the specification of Algorithm 4 and capturing the final value returned.

Definition 139 (Function to Execute the G-algorithm (Algorithm 4)):

$$f_{\text{g-execute}} : G \rightarrow \Theta \ni$$

$$\forall g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G,$$

if π is the optimal solution for problem scenario γ as returned from a run of the G-algorithm (Algorithm 4 (page 53)) against the Type-G problem g ,

$$\text{then } f_{\text{g-execute}}(g) = \pi.$$

Note the implicit dependency on the computing resources: the quality (that is, the fitness) of the output result π will, in general, be improved through the use of a more powerful computing platform.

11.4. The G^\sharp Problem

Having described the general problem domain of Type G and an algorithm by which to find optimal solutions for Type-G problems, we proceed along the line of deliberation indicated in the introduction (Subsection 11.1.1 (page 49)). That is, we proceed to describe methods, algorithms, and processes by which

1. a Type-G problem may be optimized (thus assuring the maximum performance of the G-algorithm itself), and by which the Type-G problem of doing *this* can be optimized, and, again, by which the Type-G problem of doing *that* can be optimized, etc., indefinitely (which is referred to as the G^\sharp problem), and
2. a mechanism (the G^\sharp algorithm) can be derived by which all of the problem scenarios in a given Type-G problem domain may be solved with maximum efficiency.

These additional optimization methods depend on the concept of a *Type-G meta problem*.

11.4.1. Regarding Type-G Meta Problems

Importantly for our purposes, the G-algorithm specification (Algorithm 4 (page 53)) applies not only to finding an optimal solution for a given target problem of Type G, but also to optimizing the Type-G problem itself. Suppose that, by Definition 136 (page 53),

$$g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G$$

is a Type-G problem of finding an optimal solution for a given problem scenario γ in target problem domain δ . Then there is a “meta” problem domain δ' and a “meta” problem scenario $\gamma' \in \delta'$ for finding the optimal choice of the values of the internal parameters of g , and this problem is solved as another Type-G problem

$$g' = (\delta', \gamma', \nu', \tau', \alpha', f'_r, f'_{\text{test}}, f'_m, f'_c, \pi') \in G$$

The remaining elements of the tuple g' are to be consistent with Definition 136 and must satisfy certain additional conditions that will be described below.

11.4.2. Meta-Problem Scenarios Compounded Indefinitely

Consider (as suggested above) an infinite sequence of Type-G problems whose first element is g , a Type-G problem of deriving the optimal solution of some Type-G problem scenario γ , where each element after the first element of the sequence is formed as the Type-G problem of optimizing the choice of the values of the internal parameters of the preceding Type-G problem in the sequence. This idea, designated as the G^\sharp problem, will, as in the S^\sharp problem, entail evolutionary search technology (genetic algorithms) by which to reach solutions.

Given a Type-G problem g in the infinite sequence mentioned above, the successor Type-G problem, g' , represents the task of solving the Type-G meta problem of optimizing the choice of the values of its predecessor’s internal parameters. The problem scenario for g' would be a data structure that stipulates the requirements and constraints applicable to the solution of the Type-G meta problem (and this data structure would be in one-to-one correspondence with the data structure that represents the internal parameters of g). One possible such requirement/constraint would be that the value of a , some given internal parameter of g , should fall in some particular range ($a > 10$, for example). Meta-problem scenarios and internal parameters is the subject of the next subsection.

We might contemplate a method by which to address the entire composite problem that consists of (a) optimizing the solution of the initial Type-G problem and (b) optimizing the solution of the meta problems in the infinite sequence mentioned above. In pursuing this, we would naturally confront the question of whether it makes sense to try to apply such a search algorithm or method indefinitely to an infinite sequence of problems of Type G as describe above.

11.4.3. Approaching the G^\sharp Problem

This is moot, however, since in fact we seek not a theoretical solution for this entire indefinitely compounded problem,

but rather a feasible means of attacking a useful part of it. We alluded to a similar issue in the last paragraph of Section 6.1 (page 37), and indicated that, in avoiding a kind of infinite regression, a justifiable and workable approach—

1. would first generate a set of data cases for the problem represented by the first element $g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G$ of the sequence. Each of the data cases would consist of
 - (a) an actual or realistic problem scenario $\gamma' \in \delta$ and
 - (b) the calculated optimal choice of the values of the internal parameters of the Type-G problem for solving the problem scenario γ' , where the optimal choice is discovered via an appropriate probabilistic search strategy (i.e., an evolutionary search strategy), and
2. would then use a hypersurface-fitting method to fit to the generated data cases an estimation function that, given an arbitrary problem scenario γ' belonging to the Type-G problem domain δ , would return an estimate of the optimal choice of the values of the internal parameters of $g' = (\delta, \gamma', \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G$.

The first part of the above approach is satisfied through the application of appropriately defined instances of a search algorithm—which will be specified below as an evolutionary search algorithm. The second part of the above approach is satisfied through a generalized version (the G^\sharp algorithm (see Subsection 11.5 (page 58))) of the methods and algorithms described in Section 7 (page 40). (See Revisions and Changes Digest item 27, page 67.)

11.4.3.1. Internal Parameters of Type-G Problems

Each internal parameter

$$\alpha_i, i \in \{1, \dots, \text{len}(\alpha)\}$$

of Type-G problem $(\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi)$ has an integer value that represents the number of new candidate solutions that will be added to the working population in each step respectively in each iteration of the steps listed in the definition of a G-algorithm (Algorithm 4 (page 53)). Since a problem scenario γ' for meta problem δ' is a vector of constraints on the solutions of the meta problem (that is, constraints on the values of the internal parameters of the given Type-G problem), it may, without loss of generality, be assumed that all problem scenarios for all meta problems are identical, and further that every such constraint is simply that the value of the internal parameter must be nonnegative. In fact, this assumption will align with the proposition stated elsewhere that the constraints represented by a problem scenario should be as loose as possible so as to ensure the most thorough possible exploration of the solution space by the evolutionary search algorithm.

11.4.3.2. Constructing Type-G Meta Problems We now define a function that, given a problem scenario γ in Type-G problem domain δ , returns the Type-G meta-problem scenario γ' in Type-G meta-problem domain δ' , i.e., the problem of finding an optimal choice of the values of the internal parameters of g (the Type-G problem of finding an optimal solution for the given problem scenario γ).

Definition 140:

$$\begin{aligned} F_{\text{new-g}}: \Theta \times G &\rightarrow G \ni \\ \forall (\theta \in \mathbb{N}^{\text{len}(\alpha)}, g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi)) &\in \\ \Theta \times G, & \\ F_{\text{new-g}}(\theta, g) &= (\delta, \gamma, \nu, \tau, \theta, f_r, f_{\text{test}}, f_m, f_c, \pi). \end{aligned}$$

$F_{\text{new-g}}$, given a Type-G problem g and a vector θ representing a choice of the values of the internal parameters of g , returns a new Type-G problem identical to g except that g 's internal parameters are replaced with θ . Note that Θ , the set of all permissible solutions for problems of Type G, includes permissible solutions for Type-G meta problems, and thus, a vector $\theta \in \mathbb{N}^{\text{len}(\alpha)}$ representing a choice of the values of the internal parameters of g , and representing, therefore, a permissible solution of the Type-G meta problem, belongs to Θ .

Definition 141:

$$\begin{aligned} F_{\text{META}} \subseteq G^G &\ni f \in F_{\text{META}} \Rightarrow \\ \forall g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) &\in G, \\ g' = (\delta', \gamma', \nu', \tau', \alpha', f'_r, f'_{\text{test}}, f'_m, f'_c, \pi') &= f(g) \Leftrightarrow \end{aligned}$$

1. δ' is the meta problem of optimizing the Type-G problems for solving problem scenarios in δ ,
2. $\gamma' \in \delta'$ is the meta-problem scenario of optimizing g ,
3. $\tau' = \tau$,
4. $f'_r \in F_{\text{randomselection}}(\delta', \gamma')$,
5. $f'_{\text{test}} = f_{\text{test}} \circ f_{\text{g-execute}} \circ F_{\text{new-g}}$,
6. $f'_m \in F_{\text{mutation}}(\delta', \gamma')$,
7. $f'_c \in F_{\text{crossover}}(\delta', \gamma')$.

A function f that belongs to the set F_{META} will accept as input a Type-G problem g and return as output another Type-G problem for solving the meta problem of finding an optimal choice of the values of the internal parameters of g .

A number of observations apply to the above definition in relation to its complexity, particularly as to its use in optimizing a G^\sharp problem as described below (Subsection 11.4.5). The complexity arises mainly from the definition of the fitness function f'_{test} , which is the composition⁷ of three other functions. In the Type-G meta problem g' , the fitness function f'_{test} is defined as g 's fitness function, f_{test} , applied to the

⁷ In mathematics, “function composition” (indicated by the symbol \circ) refers to the use of the output of one function as the input for another. See Definition 14 (page 19).

optimal solution obtained by executing Algorithm 4 against the modified version of g , i.e., the version of g obtained by replacing the existing choice of the values of g 's internal parameters, α , with θ , the given candidate choice of the values of the internal parameters of g . Thus, since the execution of Algorithm 4 against the modified version of g takes place during the process of running the fitness function of the Type-G meta problem, this fitness function does the heavy lifting in executing Algorithm 4 against a Type-G meta problem, as will be explained in Subsection 11.4.5.

While the insight gained in Section 10.5 (page 49) into the effect of changing the value of any given internal parameter may be relevant to Type-G problems, we do not pursue the confirmation of the relevance by modeling the performance of the G-algorithm (Algorithm 4). Instead, it can be noted that a direct approach would consist of the immediate application of some $f \in F_{\text{META}}$, where, $\forall g \in G$, the optimal choice of the values of the internal parameters of g is simply

$$f_{g\text{-execute}}(f(g))$$

An extension of this approach will be described in Subsection 11.4.5 relative to a chain of Type-G meta problems.

11.4.4. The G^\sharp Problem: A Chain of Type-G Meta Problems

We proceed to consider the generalization of the concepts addressed in the main body of this paper, namely, the problem of (1) how to choose the values of the internal parameters of the Type-G problem to maximize the performance of the G-algorithm in a fielded system, (2) how to optimize the internal parameters of the Type-G problem of optimizing *that* Type-G problem, (3) how to optimize the Type-G problem that does *this*, etc., indefinitely. We refer to this problem as the G^\sharp problem. Note that its rather abstract underlying idea will not preclude a treatment of the G^\sharp problem supporting practicality and feasibility.

Definition 142 (G^\sharp Problem):

$$G^* \subseteq \Xi^*(G) \ni \xi \in G^* \Rightarrow$$

1. $\exists g \in G \ni g = \xi_0$,
2. $\exists f \in F_{\text{META}} \ni \forall j \in \{1, \dots, \text{len}(\xi) - 1\}, \xi_j = f(\xi_{j-1})$

G^* is the set of all possible G^\sharp problems. Each element except the first element of the sequence $\xi \in G^*$ is a Type-G meta problem of optimizing its predecessor. ξ_0 , the first element of ξ , is a Type-G problem representing the *base case* for the recursive process that takes place when one of its successors in ξ is solved. By Definition 136, ξ_0 represents some Type-G problem $g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G$ for finding an optimum solution for problem scenario $\gamma \in \delta$.

11.4.5. Optimized G^\sharp Problem

Definition 143 (Optimized G^\sharp Problem):

A G^\sharp problem $\xi \in G^*$ is said to be *optimized* if and only if Algorithm 4 has been run against the Type-G meta problem $\xi_{\text{len}(\xi)-1}$, resulting, recursively, in the replacement, in Type-G problems $\xi_j, j \in \{0, \dots, \text{len}(\xi) - 2\}$, of the values of their internal parameters with the optimal values computed by recursively executing Algorithm 4 against each element of ξ starting with its last element ($\xi_{\text{len}(\xi)-1}$).

11.4.5.1. Explanation of Recursive Optimization in the G^\sharp Problem

Algorithm 4 is recursive by virtue of the definition of the fitness function of a Type-G meta problem (see remarks under Definition 141). This fact enables the optimization of the G^\sharp problem, where the entire chain (i.e., sequence) of Type-G meta problems described in Definition 142 will be optimized by application of the algorithm to the last element of the chain.

By way of further explanation, note that some prescribed function $f \in F_{\text{META}}$, given some element ξ_i in the chain (sequence) ξ as input, will have produced the Type-G meta problem

$$\xi_{i+1} = (\delta', \gamma', \nu', \tau', \alpha', f'_r, f'_{\text{test}}, f'_m, f'_c, \pi')$$

for optimizing ξ_i . By Definition 141, $\gamma' \in \delta'$ is the meta-problem scenario of optimizing target Type-G problem ξ_i . Applying Algorithm 4 to ξ_{i+1} will take the following course:

1. Step 1 executes the random-selection function f'_r and returns Π , a set of candidate solutions of problem scenario γ' , the problem of optimizing the choice of the internal parameters of Type-G problem ξ_i . Thus, Π is a set of vectors representing possible such choices, i.e., solutions of problem scenario γ' .
2. Step 2a executes the function W_{steps} with the ordered pair (Π, ξ_{i+1}) as input and then sets Π' to the value returned. As W_{steps} executes, the composite function f'_{test} will be used to test the candidate solutions individually. Since by Definition 141, f'_{test} executes the function $f_{g\text{-execute}}$ with the Type-G problem ξ_i as the input, then, by Definition 139, Algorithm 4 will itself be invoked again part-way through the steps of the self-same algorithm, thus resulting in recursion.

Step 2 of Algorithm 4 progresses through successive generations of candidate solutions, ultimately ending with either the expiration of the allowed run time ν' or the discovery of a “good enough” solution according to the parameter τ' , and the best solution found during this iterative search process is returned at the termination of step 2.

It may be helpful to elaborate somewhat on the use, in the definition of the G^\sharp problem (Definition 142), of the function

$f \in F_{\text{META}}$, which, in constructing ξ , produces, for each element ξ_i , the Type-G meta problem ξ_{i+1} that will be solved via Algorithm 4. The key part of the definition of $f \in F_{\text{META}}$ is the composite function f'_{test} . When invoked during the execution of the function W_{steps} against the set Π of candidate solutions, f'_{test} first applies the function $F_{\text{new-g}}$ to the candidate Type-G problem given as input along with the candidate solution (a choice, π (see step 2b of Algorithm 4), of the values of the Type-G problem's internal parameters), producing as output a modified version of the target Type-G problem, with π in place of the original vector α of internal parameters. This output from $F_{\text{new-g}}$ (i.e., the modified version of the target Type-G problem) is used next (according to the definition of f'_{test} as the composite function $f_{\text{test}} \circ f_{\text{g-execute}} \circ F_{\text{new-g}}$) as input to the function $f_{\text{g-execute}}$, whose output (the optimal solution returned from running Algorithm 4 against the modified version of the target Type-G problem) is used as input to the fitness function f_{test} , which is the fitness function of the target Type-G problem itself.

But the recursive nature of Algorithm 4, when used on a member of a chain of Type-G meta problems, means that the above sequence of operations continues until the *base case* is reached in the chain (i.e., the first element of the chain), where the target Type-G problem, by the definition of a G^\sharp problem (Definition 142), is *not* a Type-G meta problem (i.e., the problem scenario is not to optimize a predecessor in ξ) and, in completing the recursive process, the fitness of the solution returned from applying Algorithm 4 against it is used in following in reverse order the progression of operations that took place in the recursive chain ending at the base case.

11.4.5.2. Limits on Run Time Optimizing a G^\sharp problem $\xi \in G^*$, involving recursion during the execution of Algorithm 4, in which each new recursive step involves the full evolutionary search process that itself is a repetitive process of evolving a population of solutions through perhaps many generations, is easily seen to be very demanding computationally—the more so the longer the sequence ξ .

From an implementer's point of view, an early question would concern how much run time to allow for the optimization of ξ (see Definition 143). That is, suppose $i \in \mathbb{N}^+$ is the index of a Type-G meta problem in ξ , and

$$\xi_i = (\delta', \gamma', \nu', \tau', \alpha', f'_r, f'_{\text{test}}, f'_m, f'_c, \pi')$$

Then what value should ν' (i.e., $\xi_i[3]$) have?

Experience with an evolutionary-search application for a given problem domain will eventually teach the practitioner the empirical fact that runs of the application will process some typical number k_{gen} of generations before reaching the optimal solution. For argumentation purposes, we assume that the idea of the existence of such a typical value applies

in the case of a G^\sharp -problem optimization. Thus, if

$$\xi_0 = g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G$$

(where ν , the run time of g (or at least its typical value), will also be learned from experience) then the run time expected for the execution of the Type-G meta problem ξ_i will be given by the relation

$$\nu' = k_{\text{gen}}^i \nu \quad (9)$$

Hence, given the simplifying assumption that each evolutionary search will (with fixed computational resources) process k_{gen} generations before reaching the optimal solution, the conclusion is that the computational burden increases exponentially with the size of the G^\sharp problem $\xi \in G^*$ (i.e., exponentially with the length of ξ).

The assumption regarding k_{gen} could be modified to reflect that undoubtedly the “typical” number of generations for the runs of the application program varies as a function of the index i into the sequence ξ . That is, $k_{\text{gen}}: \mathbb{N} \rightarrow \mathbb{N}$. If we now consider that the allowed run time is a sequence ν (where $\nu_i = \xi_i[3]$) we can restate Equation 9:

$$\nu_{i+1} = k_{\text{gen}}(i+1)\nu_i \quad (10)$$

(See Revisions and Changes Digest item 29, page 67.)

Despite the fact that no principles support the *a priori* choice of an optimal vector of the values of the internal parameters of the base case (i.e., for the Type-G problem ξ_0), we must allow for the possibility that Type-G meta problems may all have the same optimal vector of the values of their internal parameters. In this circumstance, after computationally establishing this optimal vector for ξ_1 , it would render unnecessary any further computation to optimize any of the subsequent Type-G meta problems in ξ . Therefore, the question of run-time limits becomes moot in this circumstance, but whether this circumstance actually prevails in general would have little chance of resolution short of experimentation, and so in the meantime it must be categorized as speculative.

Such experimentation could hardly be conducted in the first place upon any assumption other than the one represented by Equation 10 (absent any theoretical or *a priori* knowledge to the contrary) and thus is established the usefulness of the foregoing analysis.

11.4.6. Practical Stopping Point in the G^\sharp Algorithm

As indicated previously (Subsection 11.4.3 (page 54)), however, the purpose of having an optimized G^\sharp problem is not to have some lengthy sequence of Type-G meta problems optimized, for which the necessary computational power would (as shown in the foregoing paragraphs) increase exponentially with the length of the sequence. Indeed, reflecting the author's judgment, the requirement for practical operational

use would be much more modest (see further remarks in Paragraph 11.5.1.3), and would afford adequate advantage with the sequence ξ having length 2, with a single Type-G meta problem ξ_1 to optimize the Type-G problem ξ_0 for solving a given target problem scenario. Thus, executing Algorithm 4 against Type-G problem ξ_1 would optimize the internal parameters of the Type-G problem ξ_0 , supporting the practical approaches to be described next for the use of the above $G^{\#}$ method and algorithm.

11.5. Overall Optimization: $G^{\#}$ Problem

11.5.1. The Final Issue

The final issue to be addressed relative to the generalization of the concepts covered in the main body of this paper pertains to the practicality of the $G^{\#}$ method and algorithm for operational use.

11.5.1.1. Internal Parameters Estimation Functions

Various relevant considerations were presented in Section 7 (page 40) in discussing approaches for obtaining efficient estimation functions. In the present context, an estimation function, given a problem scenario, would return an *estimated optimal choice* of the values of the internal parameters of the Type-G problem for solving the given problem scenario. (In the earlier discussion, the context was the NASA space-data communications scheduling problem, whereas here we are in a more abstract discussion of optimization of a Type-G problem.)

11.5.1.2. Concerning the Goal As in Section 7 (page 40), our goal in the present section is to maximize the practicality of the overall general optimization methods implemented in fielded systems. We address the problem, which we will refer to as the $G^{\#}$ problem, of devising a means for efficiently estimating an optimal choice of the values of a Type-G problem's internal parameters for any given new problem scenario so that it will not be necessary each time to use the $G^{\#}$ algorithm to perform the whole iterative (and computationally expensive) process of solving the internal parameter optimization problem. Clearly, the overall optimization problem is computationally very demanding, and consequently the method and algorithm described above for solving the $G^{\#}$ problem would not be advantageous to use operationally for every new problem scenario. (To be sure, such a burdensome strategy would quickly be seen to be counterproductive, since an optimal solution for the given problem scenario would be reached anyway during the very first iteration of the effort to solve the $G^{\#}$ problem, thus obviating any additional effort in that direction.) However, the $G^{\#}$ method and algorithm will support

the goal of solving the $G^{\#}$ problem of deriving an estimation function that, given an arbitrary problem scenario for the given target problem domain $\delta \in \Delta$, would, at low cost, return an estimated optimal choice of the values of the internal parameters of the Type-G problem for solving that problem scenario. The objective, then, is to specify an estimation method/algorithm that uses (abstracted and computationally inexpensive) information about the given problem scenario itself.

11.5.1.3. A Judicious Stopping Point Although the method and algorithm specified above for addressing the $G^{\#}$ problem $\xi \in G^*$ (see Definition 142) allowed for any arbitrary number of Type-G meta problems listed in the sequence ξ , the author considers (as indicated in Subsection 11.4.6) that, for practical operations using the methods and algorithms disclosed herein, the length of ξ would be 2, so that Type-G problem ξ_1 (i.e., the last element of ξ) would be the Type-G meta problem for optimizing ξ_0 for solving the initial problem scenario $\xi_0[2]$ (note that ξ_0 is a tuple, whose second element, $\xi_0[2]$, is a problem scenario γ). Indeed, upon practical considerations of computational feasibility as opposed to purely theoretical considerations, it is unlikely that any value greater than 1 for the index into ξ could be entertained at all, and so in this sense the $G^{\#}$ algorithm is academic. In the discussion that follows, it will be included in the process for implementing the $G^{\#}$ algorithm, with the understanding that, in use, there would be only one level of Type-G problem optimization (corresponding to a single application of Type-G problem optimization) representing a kind of minimal use (and perhaps the only feasible use, as will be explained) of the $G^{\#}$ algorithm).

The question of the efficiency of the Type-G problem-optimization process would itself involve the question of how to set the values of the internal parameters of ξ_1 , the Type-G meta problem for solving the meta-problem scenario $\xi_1[2]$. This is an open question, although, since all of the Type-G meta problems have identical problem scenarios (i.e., all of the constraints they specify are, by assumption (see Paragraph 11.4.3.1), the same), it cannot be dismissed that (as discussed in Paragraph 11.4.5.2) the optimal choice of the values of the internal parameters of all of the Type-G meta problems would be the same or at least nearly the same. Whether a substantial improvement in performance might occur for small changes in the values of the internal parameters is not known, but seems unlikely over a wide range of possible choices; in other words, performance may be insensitive to the choice. In that case, the effort to optimize the Type-G meta problems listed in ξ would have a correspondingly low practical justification. As to what should be the actual (optimal) choice of the values of the internal parameters of a

Type-G meta problem, there would be little *a priori* guidance that could be offered to a developer beyond that afforded by Appendix A (page 44) combined with trial-and-error experience. (See Revisions and Changes Digest item 28, page 67.)

11.5.1.4. Challenges From the Real World It should be well noted that real-world problem domains of Type G typically entail significant complexities that inevitably would translate into challenges in using the methods presented in this appendix. An estimation function as described above, even if derivable in principle, may be difficult to obtain within realistic limits on computing resources (see also the remarks in Section 7.4 (page 41) and Subsection 11.4.6 (page 57)).

The approaches to be described for solving the $G^{\#\#}$ problem will produce results (i.e., will produce instances of estimation functions), but the approaches assume a selection of real-world input data to which, in effect, a model will be fitted (see the outline of the approach given in Subsection 11.4.3 (page 54)). The accuracy of the derived estimation functions will be directly related not only to the shrewdness of the choices that determine the form or architecture of the model, but also to the selected input data (the training data) to which the model will be fitted, i.e., the number of precomputed data points and their distribution across the solution space. The required accuracy of the derived estimation functions may dictate a significant number of precomputed data points, and correspondingly significant computing resources [24].

Further, since the accuracy metric undoubtedly would exhibit asymptotic behavior in relation to the number and distribution of precomputed data points, and since the asymptote is not known in advance, even more computation would be needed to gain the assurance of any prescribed accuracy. Thus, the real-world challenges in using the methods to be described would be considerable. However, the methods can be expected to achieve the objective mentioned in Paragraph 11.5.1.2 (page 58) given a reasonably tame relationship between the independent variable (the problem scenario (or rather its characterization (see Paragraph 11.5.1.5))) and the dependent variable (the optimal choice of the values of the internal parameters of the Type-G problem for solving the problem scenario). The tameness of this relationship may be expected to be target-problem dependent, but an independent means of determining the tameness in advance is unknown—although, reasonably, some probabilistic method would be strongly indicated (if not unavoidable).

11.5.1.5. Characterization Functions The approaches described in Section 7 relied upon the concept of a problem-scenario characterization function—which is a con-

cept also involved in the approaches described in the following paragraphs concerning mechanisms for estimating optimal choices of the values of the internal parameters of Type-G problems. The characterization function is intended to be a computationally inexpensive means of distinguishing between problem scenarios in terms that are relevant to the efficiency of the evolutionary search methods that will be used in solving arbitrary problem scenarios in the given Type-G problem domain. The efficiency of the search methods that we disclose herein is related to (among possibly many other things) the sizes of the data structures that represent the elements of the problem scenario: by Definition 123 (page 50), a problem scenario is a finite sequence each element of which is a finite data structure. In the case of the problem domain of scheduling space-data communications (see Section 7 (page 40)), each of these data structures was a set, and the sizes of all of the data structures listed in a given problem scenario were used as the elements of the vector returned by the characterization function (Definition 122 (page 40)). Partly for reasons of tractability and operational efficiency, we elect to use a similar scheme here.

We proceed by specifying, in the context of Type-G problem domains, the concept of a problem scenario-characterization function, noting from Definition 128 (page 50) that, for a given target problem domain δ of Type G, there exists a positive integer $k_p = \Delta_{\dim}(\delta)$ such that all problem scenarios $\gamma \in \delta$ have length k_p , and, thus, every problem scenario $\gamma \in \delta$ has a k_p -dimensional characterization

$$c \in \mathbb{N}^{k_p}$$

For the $S^{\#\#}$ problem, the characterization was defined so that $\forall j \in \{1, \dots, k_p\}, c_j$ was the size of the finite data structure γ_j (see Definition 122 (page 40)). For the generalized problem of Type G, the definition of the characterization functions must be somewhat generalized and must accommodate Type-G meta problems.

Definition 144 (Problem-Scenario-Characterization Function for Given Problem Domain):

$$\Lambda: \Delta \rightarrow \wp(D_0^{\Delta \times \Gamma}) \ni \\ \forall \delta \in \Delta, \lambda \in \Lambda(\delta) \Rightarrow \forall \gamma \in \delta, \lambda(\delta, \gamma) \in \mathbb{N}^{\Delta_{\dim}(\delta)}$$

$\Lambda(\delta)$ returns a set of characterization functions for the problem domain δ .

11.5.2. Regression Methods for Solving the $G^{\#\#}$ Problem

Deriving an estimation function $f_{\text{estimation}}$ that returns an estimate of the optimal choice of the values of the internal parameters of a Type-G problem may be accomplished through the application of some form of *regression technology* [16].

(Use of the term “regression” herein is intended to encompass the general category of regression-analysis technologies.) (See Revisions and Changes Digest item 32, page 67.)

The estimation function to be derived is imagined (with the risk of oversimplification) as a hypersurface whose domain (consisting of the values of the independent variable) is the set of all possible problem scenarios (or, rather, their characterizations) and whose codomain (consisting of the values of the dependent variable) is the set of estimated optimal choices of the values of the internal parameters of the Type-G problems for solving problem scenarios in target problem domain δ . An applicable modeling approach as identified above represents a kind of hypersurface-fitting technique, as mentioned in the discussion in Subsection 7.2 (page 40), which concerned a method for scheduling-algorithm optimization.

It should be reiterated that applying any regression technology would be subject to cautions similar to the ones stated earlier (Subsection 7.4 (page 41) and Paragraph 11.5.1.4 (page 59)).

Regardless of which regression technology is employed, the derived estimation function would have value in an operational system in terms of maximizing the performance of the G-algorithm. For any given problem scenario to be solved by the system, the values of the internal parameters of the Type-G problem would first be adjusted according to the estimate that would be generated by the derived estimation function. The process for using the derived estimation function operationally will be further specified in Subsection 11.7 (page 64).

The objective of applying a regression technology to a $G^{\#\#}$ problem is to derive an estimation function $f_{\text{estimation}}$ from a given known data set $Q \subseteq D_0^2$, where for each $(c, e) \in Q$, c is a characterization of a problem scenario and e is a corresponding calculated optimal choice of the values of the internal parameters of the Type-G problem for solving the problem scenario.

11.5.3. Functions That Model a Given Data Set

It is assumed that a regression-analysis technology will have been selected for application in solving $G^{\#\#}$ problems. Solving a given $G^{\#\#}$ problem would consist of applying the selected regression analysis technology to an appropriate pre-computed data set Q to derive a function that models Q . The derived function may then be used to estimate the value of the dependent variable for any given value of the independent variable. The value of the independent variable for a given $G^{\#\#}$ problem would be a tuple representing the characterization of the given problem scenario, and the value of the dependent variable would be a tuple representing the estimated optimal choice of the values of the internal param-

eters of the Type-G problem for solving the given problem scenario.

Definition 145 (Set of Regression Analysis Technologies):

$$A_r = \{a: a \text{ is a regression-analysis technology}\}.$$

It will be understood that regression-analysis technology $a_{\text{regr}} \in A_r$ has been selected for use in solving $G^{\#\#}$ problems.

Definition 146 (Functions That Model a Given Data Set):

A function $f \in D_0^{D_0}$ is said to *model* $Q \subseteq D_0^2$ if and only if f has been derived through the application of regression-analysis technology a_{regr} to Q and

$$(c, e) \in Q \Rightarrow f(c) = e$$

Note that f in the above definition is a subset of D_0^2 . (See Revisions and Changes Digest item 30, page 67.)

Definition 147:

$\Upsilon: \wp(D_0^2) \rightarrow D_0^{D_0} \ni \forall Q \subseteq D_0^2, \Upsilon(Q)$ is a model of Q derived by applying regression-analysis technology a_{regr} to the data set Q .

Thus, if $\Upsilon(Q)$ models Q , then $\forall (c, e) \in Q, \Upsilon(Q)(c) = e$.

Definition 148 (Training-Data Sets):

$$\begin{aligned} D: \Delta \times \wp(\Gamma) \times \mathbf{codomain}(\Lambda) &\rightarrow \wp(\wp(D_0^2)) \ni \\ \forall (\delta, B \subseteq \delta, \lambda \in \Lambda(\delta)) \in \Delta \times \wp(\Gamma) \times \mathbf{codomain}(\Lambda), \\ \forall Q \in D(\delta, B, \lambda), (c, e) \in Q &\Rightarrow \\ \exists \gamma \in B \text{ and } \exists \text{ optimized } G^{\#} \text{ problem } \xi \in G^* &\ni \end{aligned}$$

1. $\xi_0 = (\delta, \gamma, \bullet, \bullet, \alpha, \bullet, \bullet, \bullet, \bullet) \in G_{\text{scenario}}(\delta, \gamma)$,
2. $c = \lambda(\delta, \gamma)$, and
3. $e = \alpha = \xi_0[5]$

(See Revisions and Changes Digest item 31 (page 67).)

The function D returns the set of all possible training-data sets for $G^{\#\#}$ problems (to be described in the next section). Each such data set Q will be assumed to comprise actual calculated data. Each member of Q , then, is an ordered pair (c, e) , where c is the calculated characterization of some realistic/actual problem scenario drawn from the given set B , and, for a corresponding $G^{\#}$ problem ξ that has been optimized by running Algorithm 4 (page 53) (see Definition 143 (page 56)), e is the optimal choice of the values of the internal parameters of the Type-G problem ξ_0 . It is assumed that each such training-data set Q can be modeled by applying to Q the selected regression-analysis technology a_{regr} . The derived model is identified as an estimation function that, given the characterization of an arbitrary problem scenario γ in the given problem domain δ , will return the estimated optimal choice of the values of the internal parameters of the Type-G problem for solving γ .

11.5.4. The $G^{\#\#}$ Problem

Definition 149 ($G^{\#\#}$ Problem):

A tuple

$$\left(\delta \in \Delta, B \subseteq \delta, \lambda \in \Lambda(\delta), f_{\text{estimation}} \right)$$

is said to be a $G^{\#\#}$ problem if and only if

1. each member of B is an actual/realistic problem scenario,
2. $\exists Q \in D(\delta, B, \lambda) \ni f_{\text{estimation}} = \Upsilon(Q)$.

11.5.5. The $G^{\#\#}$ Algorithm Employing Regression-Analysis Technology

The essential steps in applying a regression-analysis approach in the $G^{\#\#}$ problem are as follows:

Algorithm 5 ($G^{\#\#}$ Algorithm Employing Regression Analysis):

Given:

- $\delta \in \Delta$,
- $B \subseteq \delta$ is a set of actual/realistic problem scenarios selected from the set δ ,
- $\lambda \in \Lambda(\delta)$.

(Note: The results of running an implementation of the present algorithm are highly dependent on the number and distribution of the scenarios in the training-data set B . If the accuracy of the estimation function generated by the implementation is not deemed adequate, then these scenarios would need to be revised to improve their solution-space coverage and used in a fresh rerun.)

Perform the following steps:

1. Let $Q \in D(\delta, B, \lambda)$.
2. Perform regression analysis using the training-data set Q , resulting in the determination of a function that best fits the members of Q . That is, let $f_{\text{estimation}} = \Upsilon(Q)$.
3. If step 2 (the regression-analysis step) succeeds, then form the $G^{\#\#}$ Problem

$$(\delta, B, \lambda, f_{\text{estimation}})$$

and exit indicating success. If regression analysis fails, then exit indicating failure, calling upon the user to alter the given set B of actual/realistic problem scenarios (e.g., by increasing their number or variety) (noting that this alteration gives an altered $G^{\#\#}$ problem) and re-run the algorithm.

Step 1 above will, in general, involve significant computation of a recursive nature using Algorithm 4 (page 53) to derive the data set Q (see Definition 148 (page 60)).

11.6. Implementation Process

Implementation of the generalized methods and algorithms described in this Appendix may be straightforward (not to say trivial) for some problem domains but likely would be challenging for others. For the sake of completeness of this disclosure, we now delineate a nominal implementation process, which would be applicable to all implementation efforts, but which does not preclude appropriate variations or adaptations for particular cases.

11.6.1. The Basic Implementation Alternatives

The minimum implementation effort would have the goal of building a system that implements a Type-G problem

$$g = \left(\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi \right) \in G$$

configured initially with arbitrary (non-optimized) choices of the values of the internal parameters α for solving the given problem scenario $\gamma \in \delta$. A more ambitious possible effort would have this minimal implementation as one goal, and would have the further goal of implementing a system for solving the $G^{\#\#}$ problem for the same problem domain $\delta \in \Delta$.

In either case, the implemented system will produce optimal solutions for problem scenarios in the target domain δ . The system developed in the more ambitious possible effort mentioned above would be expected to perform more efficiently in the routine operations mode—at the cost of the effort to solve the $G^{\#\#}$ problem to derive the estimation function (see Subsection 11.5) for estimating, for each given problem scenario γ , the optimal choice of the values of the internal parameters of $g \in G_{\text{scenario}}(\delta, \gamma)$.

The general implementation approach that will be described below in detail is first to implement modules separately, and then to integrate those modules, together with all other necessary modules (e.g., interfaces with users and ancillary systems), into the overall fielded system, with appropriate testing, documentation, etc.

11.6.2. The First Implementation Alternative (Basic Implementation of the G-Algorithm)

In the first alternative (i.e., the basic implementation of the G-algorithm), the implementation steps follow the first few steps enumerated in the G-algorithm specification (Algorithm 4 (page 53)). A subsystem that is capable of creating and solving a Type-G problem

$$g = \left(\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi \right) \in G$$

would be constructed by means of the following process:

Process 2 (Process for Implementation, First Alternative (Basic Implementation for Creating and Solving a Type-G Problem)):

Given:

- $\delta \in \Delta$
- $\gamma \in \delta$

Perform the following steps:

1. Implementation steps for a subsystem for creating a Type-G problem for solving $\gamma \in \delta$.

- (a) Develop a module that supports the creation/setting of—
 - i. $\nu \in \mathbb{N}^+$, a value that represents the run-time limit in units of seconds.
 - ii. $\tau \in \mathbb{R}$, a small nonnegative value to represent a policy or judgment as to how close to perfect a solution must be (i.e., how close a solution's fitness must be to unity) to be considered “good enough” to exit the algorithm. This value normally would be small enough (0, say) to ensure that the algorithm always ran for the maximum allowed run time ν .
 - iii. $\alpha \in D_0$, the vector representing the user's initial choice of the values of the internal parameters of g , with $\mathbf{len}(\alpha) = \Delta_{\dim}(\delta)$.
- (b) Develop a module that embodies a random-selection function $f_r \in F_{\text{randomselection}}(\delta, \gamma)$
- (c) Develop a module that embodies a fitness function $f_{\text{test}} \in F_{\text{fitness}}(\delta, \gamma)$.
- (d) Develop a module that embodies a genetic-mutation function $f_m \in F_{\text{mutation}}(\delta, \gamma)$.
- (e) Develop a module that embodies a genetic-crossover function $f_c \in F_{\text{crossover}}(\delta, \gamma)$.
- (f) Develop a module that supports the creation of a place-holder data structure π representing an arbitrary member of $\Theta_{\text{scenario}}(\delta, \gamma)$.
- (g) Develop a module that supports the creation of the data structure

$$g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G_{\text{scenario}}(\delta, \gamma)$$
 according to Definition 136 (page 53) and Definition 137 (page 53)

2. Implementation steps for a subsystem embodying Algorithm 4:

- (a) Develop a module that embodies the function W_{steps} according to Definition 138 (page 53), incorporating the module developed in step 1g.

- (b) Develop a module to capture Π , the output of function f_r when given the input equal to

$$\sum_{j \in \{1, \dots, \mathbf{len}(\alpha)\}} \alpha_j$$

- (c) Develop a module to capture Π' , the output of function W_{steps} (the module developed in step 2a) when given the ordered pair (Π, g) as input, where Π and g are the outputs of the modules developed in steps 2b (for creating a random-selection function $f_r \in F_{\text{randomselection}}(\delta, \gamma)$ (for a given (δ, γ)) and 1g (for for creating a Type-G problem), respectively.
 - (d) Develop a module to find a member $\pi' \in \Pi' \ni \zeta \in \Pi' \Rightarrow f_{\text{test}}(\zeta) \geq f_{\text{test}}(\pi')$, where Π' is the set captured by executing the module developed in step 2c.
 - (e) Develop a module that sets π to the output, π' , from executing the module developed in step 2d.
 - (f) Develop a module that sets Π to the output, Π' , from executing the module developed in step 2c.
 - (g) Develop a module that performs the test $f_t(\pi) < 1 + \tau$ and tests whether the elapsed run time exceeds ν , and if either test is affirmative, then exits, returning the value π , and, if otherwise, then resumes execution of the sequence of modules developed in steps 2c through 2g.
3. Develop a module that embodies the function $f_{g\text{-execute}}$, which in turn will incorporate the module embodying Algorithm 4 developed in step 2.
 4. Develop a module that (a) executes, with $g \in G$ as input, the module embodying the function $f_{g\text{-execute}}$ developed in step 3 for a given Type-G problem and (b) returns the optimal solution of g .
 5. Perform system integration and testing of the modules developed as specified above, along with all other necessary modules (e.g., user interfaces).

Once implemented, the fielded system would be used operationally according to the following straightforward process:

Process 3 (Process for Operational Use of Implementation Under First Alternative (Basic Implementation for Creating and Solving a Type-G Problem)):

Given:

- $\delta \in \Delta$
- $\gamma \in \delta$

Perform the following steps:

1. Prepare all input data required for a run of the system developed in Process 2 (page 62) (including the target problem scenario and other inputs as required by the G-algorithm specification (Algorithm 4 (page 53))).
2. Initiate a run of the system with the stipulated inputs and capture output upon termination of run. The output is an optimal solution for the given problem scenario γ (see remarks in Section 2.4.7 on page 15 concerning the concept of “optimum”).

11.6.3. The Second Implementation Alternative (Implementation of the $G^{\#}$ Algorithm)

The second implementation alternative incorporates all of the steps in the first alternative as described above (Process 2 (page 62)), which produces an implementation of a system that can create and solve a Type-G problem

$$(\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G$$

The additional steps required in the second implementation alternative will be described next, and have the objective of maximizing the performance of the fielded system over the broad range of possible problem scenarios. This objective is achieved by implementing the $G^{\#}$ algorithm (see Algorithm 5 (page 61), which incorporates the $G^{\#}$ problem optimization (see Definition 143 (page 56)), which, in turn, incorporates the G-algorithm (Algorithm 4 (page 53)).

It is necessary, then, to describe implementation steps for the $G^{\#}$ problem optimization, as well as the $G^{\#}$ algorithm).

Process 4 (Process for Implementation, Second Alternative (Implementation of $G^{\#}$ Algorithm (Algorithm 5 (page 61))):

Given:

- $\delta \in \Delta$

Perform the following steps:

1. **Implementation Steps for Support-Modules for Type-G Problem Optimization.** Supporting modules in an implementation of a Type-G problem-optimization function will be constructed by means of the following steps:
 - (a) Develop (see implementation steps specified in Process 2 (page 62)), or incorporate, a module that supports the creation and solving of a Type-G problem

$$g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G.$$
 Note that this module incorporates (from step 3 of Process 2) a module that implements the G-algorithm-execution function $f_{g\text{-execute}}$ (see Definition 139).
 - (b) Develop a module that embodies the function $F_{\text{new-g}}$ (see Definition 140 (page 55))
 - (c) Develop a module that embodies the definition of a function in the set F_{META} that, given a Type-G problem

$$g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G$$
 returns a Type-G meta problem g' (see Definition 141 (page 55) in Paragraph 11.4.3.1).
2. **Implementation Steps for a Subsystem that Supports the Creation of a Module that Embodies the $G^{\#}$ Problem Optimization.** A subsystem that supports the creation and optimization of a $G^{\#}$ problem $\xi \in G^*$ (Definition 142) (where $\text{len}(\xi) = 2$, in accord with the position taken in Subsection 11.4.6 (page 57) and in Paragraph 11.5.1.3 (page 58)) will result from integrating modules constructed by means of the following steps:
 - (a) Develop or incorporate a module that implements the G-algorithm-execution function $f_{g\text{-execute}}$ (see step 3 of Process 2).
 - (b) Develop a module that creates the $G^{\#}$ problem $\xi \in G^*$, with $\xi_0 = g$, in accordance with Definition 142 (page 56), where

$$g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G$$
 is a given Type-G problem. This module incorporates the module developed in step 1c, which is an implementation of a function in F_{META} , which in turn incorporates the module developed or incorporated in step 2a for the function $f_{g\text{-execute}}$.
 - (c) Develop a module that incorporates the module developed in step 2 of Process 2 and supports the execution of Algorithm 4 against the Type-G problem $\xi_{\text{len}(\xi)-1}$, which results in the optimized version of the $G^{\#}$ problem $\xi \in G^*$ (see Definition 143 (page 56)).
3. **Implementation Steps for the $G^{\#}$ Algorithm Via Regression Analysis.** A system incorporating a regression-analysis approach for solving a $G^{\#}$ problem $(\delta, B, \lambda, f_{\text{estimation}})$ will be constructed by means of the following steps:

Given:

$$\delta \in \Delta$$

Perform the following steps:

 - (a) Develop a module that embodies a problem-scenario characterization function $\lambda \in \Lambda(\delta)$ (see Definition 144 (page 59)).
 - (b) Develop a module to support the creation of a training-data set $B \subseteq \delta$ consisting of actual/realistic problem scenarios in δ .

- (c) Incorporate a module supporting the creation of Type-G problems for problem scenarios for the given target problem domain δ (see step 1a above and Subsection 11.6.2).
- (d) Develop or incorporate a module that embodies the function D (see Definition 148 (page 60)), which incorporates the module developed in step 2c for optimizing a $G^\#$ problem.
- (e) Develop or incorporate a module that embodies the function Υ (see Definition 147 (page 60)), which implements the regression analysis technology a_{regr} (see explanation below Definition 145 (page 60)), and which incorporates the modules developed in steps 3a and 1a.
- (f) Develop a module that executes the module developed in step 3e and captures the output (i.e., the function $f_{\text{estimation}} = \Upsilon(Q)$, where $Q \in D(\delta, B, \lambda)$).
- (g) Develop a module that
 - i. executes the module developed in step 3a and retains the function λ .
 - ii. executes the module developed in step 3b and retains the set B .
 - iii. executes the module developed in step 3d and retains the set $Q \in D(\delta, B, \lambda)$.
 - iv. executes the module developed in step 3f and retains the function $f_{\text{estimation}} = \Upsilon(Q)$.
 - v. if the regression-analysis step 3(g)iv is successful, then constructs the $G^\#$ problem $(\delta \in \Delta, B \subseteq \delta, \lambda \in \Lambda(\delta), f_{\text{estimation}})$ and exits indicating success; otherwise, exits indicating failure and calling upon the user to start a new run, ensuring that the set B retained from step 3(g)ii is more appropriate in solution-space coverage.

4. **System Integration.** Perform system integration and testing of the modules developed as specified above, along with all other necessary modules (e.g., user interfaces) for the final fielded system.

The final result of the implementation steps given above would be a system consisting of (a) a subsystem by which a Type-G problem can be used to produce optimal solutions for any given problem scenario and (b) a subsystem by which the optimal choice of the values of the internal parameters of a given Type-G problem can be estimated for arbitrary problem scenarios to enable the most efficient overall possible operation of the system for arbitrary problem scenarios.

For reasons similar to those articulated earlier (Section 7.4 (page 41)), the implementation effort described above—which invokes regression-analysis technologies—is nontrivial and will necessarily require the involvement of experts in the selected regression technology, especially in relation to step 3e of Process 4.

11.7. Estimation Function: Operational Use

The resulting tested and verified estimation-function implementation would become a tool for operational use within a fielded system. The routine use of such a tool would involve the following process:

Process 5 (Operational Use of Estimation-Function Tool):

Given:

- $G^\#$ problem

$$(\delta \in \Delta, B \subseteq \delta, \lambda \in \Lambda(\delta), f_{\text{estimation}})$$

created by running the subsystem that represents the implementation of the $G^\#$ algorithm and solves the given $G^\#$ problem (see implementation Process 4, steps 3(g)i through 3(g)v), which results in the implementation of an estimation function $f_{\text{estimation}}$.

- a problem scenario $\gamma \in \delta$.

Perform the following steps:

1. Compute the characterization $c = \lambda(\delta, \gamma)$.
2. Run the module that creates an implementation of a G-algorithm (see Process 2 (page 62)), resulting in an implementation of $g = (\delta, \gamma, \nu, \tau, \alpha, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G$.
3. Capture the estimation function output $e = f_{\text{estimation}}(c)$ (the estimate of the optimal choice of the values of the Type-G problem's internal parameters for the problem scenario γ).
4. Configure the Type-G problem g , replacing α with e : $g_{\text{opt}} = (\delta, \gamma, \nu, \tau, e, f_r, f_{\text{test}}, f_m, f_c, \pi) \in G_{\text{scenario}}(\delta, \gamma)$. g_{opt} is the optimal Type-G problem for solving problem scenario γ .
5. Initiate a run of Algorithm 4 (the module developed in step 2 of Process 2) against the implementation of g_{opt} and capture output upon termination of run. The output is an optimal solution for the given problem scenario γ (see remarks in Section 2.4.7 on page 15 concerning the concept of "optimum").

11.8. Which Regression Analysis Algorithm?

Interesting (but not necessarily academic) future work would, for some selected problem domain, consist of deriving an optimal-parameters-estimation function using different regression-analysis technologies and comparing their execution performance. The comparison would be more meaningful and reliable if it were based on the same computing resources and the same training/test data (i.e., the same precomputed data cases (see definition of the function D (Definition 148), which specifies the set of all possible precomputed data sets for a given $G^{\#}$ problem)).

11.9. Final Remarks

This appendix has described a generalization of the methods and algorithms that were specified in the main body of this paper, which targeted a specific problem domain (the space-data communications scheduling problem). These generalized methods and algorithms are applicable to any problem in the very large class of real-world problems that are represented by problem domains of Type G (see Definition 127 (page 50) and Subsection 11.1.3). The specifications of the generalized methods and algorithms are sufficiently rigorous and complete to support their implementation (as described in Section 11.6 (page 61)) as a fielded system that efficiently produces an optimal solution for any problem scenario in any target problem domain of Type G.

The $G^{\#}$ problem $\xi \in G^*$ (see Definition 142 (page 56)), i.e., the problem of optimizing the Type-G problems and Type-G meta problems themselves, was incorporated into the $G^{\#\#}$ problem (i.e., the problem of devising an estimation function by which to obtain, for an arbitrary problem scenario, an estimate of the optimal choice of the values of the internal parameters of the Type-G problem g —it being noted that Algorithm 4 (page 53) applied to the Type-G problem g solves (i.e., produces an optimal solution for) the given problem scenario). For reasons explained in Paragraph 11.5.1.3 (page 58), the $G^{\#}$ -problem optimization algorithm will be applied by executing Algorithm 4 against the Type-G meta problem ξ_1 , resulting in an optimized version of the Type-G problem $\xi_0 = g$. The methods and algorithms for solving a $G^{\#\#}$ problem ($\delta \in \Delta, B \subseteq \delta, \lambda \in \Lambda(\delta), f_{\text{estimation}}$) (see Definition 149 (page 61)) invoked regression-analysis technologies as a means to fit a model (a hypersurface) to a set of known data points precomputed using the $G^{\#}$ method and algorithm, and thereby to derive the estimation function $f_{\text{estimation}}$. With adequate computing resources and talent, these overall optimization approaches, as specified, can be implemented as a fielded system that performs with maxi-

imum feasible efficiency solving any problem scenario in any target problem domain of Type G.

12. Revisions and Changes Digest

The following list identifies and recapitulates significant revisions that were included in the present document.

1. In Subsection 1.1, page 7, the last three sentences of the first paragraph replaced the following three original sentences:

Present operational scheduling capabilities do not generate true optimized schedules for reasons that will be explained below, but can (when the option is invoked) generate schedules that are free of radio frequency interference (RFI) effects by blocking out portions of the problem-solution space from consideration whenever those portions appear with any predicted RFI effects. Similarly, the current operational scheduling system prunes away portions of the solution space upon encountering violations of the various other constraints that must be satisfied. This approach, which perforce ignores large portions of the solution space, necessarily means that schedule optimization cannot be an actual achievable objective of the current scheduling system.

In the second paragraph, the first sentence replaced the following original first sentence:

Present space data-communications schedulers have the capability of algorithmically generating schedules using techniques for representing and exploring the problem-solution space as either a graph or a tree of related sub-solutions.

The last sentence of the second paragraph replaced the following original last sentence:

NASA's present operational scheduling system, using such standard search methods, is capable of producing *workable* schedules, albeit with certain significant concessions to the compute-intensive nature of the search (including certain problem simplifications that themselves, even ignoring the performance of the search techniques, preclude the possibility of true schedule optimization).

2. In Subsection 1.6, page 9, the third sentence of the last paragraph replaced the following original third sentence:

But numerous other fields (particularly those related to design optimization, and, more broadly, virtually any field where solutions cannot be specified directly but *can* be evaluated as to “goodness”) are encompassed under the generalized problem of Type G as defined in Appendix B.

3. In Subsection 2.4.8, page 16, the second sentence of the first paragraph replaced the following original second sentence:
The current NASA scheduling system, based on constructive techniques or graph-search techniques, does not incorporate a strategy for constructing solutions that will directly optimize these metrics.
4. After the first paragraph of Subsection 4.2 (page 18), two paragraphs of explanatory remarks were inserted concerning set-builder notation and the concept of logical negation.
5. On page 19, the revised, equivalent definition of \circ (“function composition”) (Definition 14) clarified the meaning of the following original definition of \circ :
 $\forall f \in X^Y, \forall g \in Z^X, g \circ f \in Z^Y$.
6. On page 19, the definition of “set of all real numbers” (Definition 7) was moved so as to precede the definition of $|\bullet|$ (cardinality of a set) (Definition 10)), thereby to avoid forward referencing.
7. On page 19, the definition of “power set” (Definition 9) was moved so as to precede the definition of $\Omega_{\mathbf{F}}$ (the set of all finite sets (Definition 10)), thereby to avoid forward referencing.
8. On page 19, the definition of $\bar{\mathbb{Z}}$, the “set of all integer intervals” (Definition 16), was positioned before the definition of **rdint** (Definition 17) and replaced the following original definition of the set of all closed intervals:
 $\bar{\mathbb{Z}} = \{i : \exists a, b \in \mathbb{Z} \ni a \leq b \text{ and } i = [a, b]\}$.
A second sentence was added to the remark below the definition of $\bar{\mathbb{Z}}$.
9. On page 20, the remark following the definition of “sequence” (Definition 21) replaced the following original remark:
Note that the first element of a sequence has index-value 0, and that no index value between the first and the last is skipped.
10. On page 20, after the definition of **len** (Definition 23), the remark was expanded and the unneeded definition of “Set of all n -tuples”, formerly in the original text, was deleted.
11. On page 21, the third line of the definition of the **max** function (Definition 31) replaced the following line of the original definition:
 \forall bounded and closed subset $Q \in \mathbb{R}, \exists x \in Q \ni$
12. On page 21, the third line of the definition of the **min** function (Definition 32) replaced the following line of the original definition:
 \forall bounded and closed subset $Q \in \mathbb{R}, \exists x \in Q \ni$
13. On page 23, the last sentence of the definition of C (Definition 53) replaced the following original last sentence:
 p is said to be a *prototype event* if and only if $\exists u \in U_0 \ni (\bullet, p) \in C(u)$.
14. On page 24, the definition of M_{type} (Definition 58) replaced the following original definition:
 $M_{\text{type}}: R_0 \rightarrow \Xi^*(\Xi^*(M)) \ni$
 $\forall r = (r_1, \dots, r_{17}) \in R_0,$
 $M_{\text{type}}(r) = \xi \in \Xi^*(\Xi^*(M)) \Leftrightarrow$
(a) $[i \in \mathbb{N}, i < \mathbf{len}(\xi)] \Rightarrow \xi_i[1] = r_2 \text{ and } \xi_i[2] = r_4$
and
(b) $[i, j \in \mathbb{N}, i < j < \mathbf{len}(\xi)] \Rightarrow$
 $[\xi_i[4], \xi_i[5]] < [\xi_j[4], \xi_j[5]]$
15. On page 24, the definition of **tref** (Definition 60) replaced the following original definition:
tref: $R_0 \times M \rightarrow \mathbb{Z} \ni$
 $\left[(r = (r_1, \dots, r_{17}), \mu = (\mu_1, \dots, \mu_5)) \in R_0 \times M, \right.$
 $\mu_1 = r_2, \mu_2 = r_4,$
 $[r_4 = \text{“NIL”} \Rightarrow t_p = r_{15}],$
 $\left. [r_4 \neq \text{“NIL”} \Rightarrow t_p = M_{\mathbf{T}}(r_8, \mu) + r_6] \right] \Rightarrow$
tref(r, μ) = t_p
16. On page 26, the definition of **skipsat**^{R*} (Definition 66) replaced the following original definition:
skipsat^{R*}: $\Theta \times R_0 \rightarrow \mathbb{R} \ni$
 $\left[(\theta, r = (r_1, \dots, r_{17})) \in \Theta \times R_0, \right.$
 $n = \mathbf{len}(M_{\text{skips}}(r, 0)),$
 $Q = \{p: \exists j, k \in \mathbb{N} \ni j < n, k < \mathbf{len}(r_3),$
 $p \in C_0^{\text{PRM}}(k, r, M_{\text{type}}(r) [M_{\text{skips}}(r, 0)[j]])\},$
 $p \in \theta\},$
 $h = \mathbf{max}(\{10^{-3}, |Q|\}) \right] \Rightarrow$
skipsat^{R*}(θ, r) = nh^{-1}
17. On page 34, the last part of the paragraph under the definition of **swappeionu** (Definition 114) replaced the following original wording:
... prototype-event instantiations for r belonging to θ' .
18. The last part of the paragraph under the definition of **swapearypeionr** (Definition 115, page 34) replaced the following original wording:
... and a randomly selected mission event instance of type r_4 all prototype-event instantiations for r earlier than m belonging to θ are swapped with all prototype-event instantiations for r earlier than m belonging to θ' .

19. The last part of the paragraph under the definition of **swapmidpeionr** (Definition 116, page 34) replaced the following original wording:
... and two randomly selected mission event instances μ and μ^* of type r_4 all prototype-event instantiations for r inclusively between μ and μ^* belonging to θ are swapped with all prototype-event instantiations for r earlier than m belonging to θ' .

20. On page 34, the definition of **rndsvcs** (Definition 117) replaced the following equivalent but less straightforward original definition:

$$\begin{aligned} \mathbf{rndsvcs}: \mathbb{N}^2 \times R_0 \times M &\rightarrow \wp(\mathbb{N} \times A_0 \times \mathbb{N}^2) \ni \\ \forall(n, k, r = (r_1, \dots, r_{17}), \mu = (\mu_1, \dots, \mu_5)) &\in \\ \mathbb{N}^2 \times R_0 \times M, & \\ (t_p, a, s, d) \in \mathbf{rndsvcs}(n, k, r, \mu) &\Rightarrow \\ \mu_1 = r_2, \mu_2 = r_4, k < \mathbf{len}(r_3), n < \mathbf{len}(r_3[k]), & \\ t_p = \mathbf{tref}(r, \mu), & \\ \left[Q = \left\{ (t_p, a, s^*, d^*) \in \mathbb{N} \times A_0 \times \mathbb{Z} \times \mathbb{N} : \right. \right. & \\ \exists \mu^V \in M \ni & \\ (t_p, a, s^*, d^*) \in Y_{\max}^I(r, k, n, \mu, \mu^V) \left. \right\}, & \\ (t_p, a, s', d') = \mathbf{rndmember}(Q), & \\ (\bullet, s^-, s^+, \bullet, d^+) = r_3[k][n], & \\ \zeta = [s', (s' + d')] \cap [s^-, (s^- + d^+)] &\Rightarrow \\ \left[s = \mathbf{rndint}(\zeta^-, (\zeta^+ - d^-)), & \\ d_{\max} = \mathbf{min}(\{d^+, (\zeta^+ - s)\}), & \\ d = \mathbf{rndint}(d^-, d_{\max}) \right] & \end{aligned}$$

21. The explanatory paragraph under Section 5 (page 35) was expanded.
22. In the original specification of Algorithm 2 in Section 6.3 on page 38, all occurrences of \bullet^* (that is, “*” used as a superscript) were deleted. This change is typographical, for the purpose of simplification only, and does not alter the meaning of the algorithm specification.
23. The last paragraph of Subsection 8.2 on page 42 is a remark added to the original text.
24. The definition of D_0 (Definition 131) replaced the following original definition of D_0 :
 $D_0 = \{x: \exists n \in \mathbb{N} \ni x \in \mathbb{N}^n\}$
and was moved to the end of Subsection 11.1.2 on page 49.
25. On page 51, item 1 in the definition of $F_{\text{randomselection}}$ (Definition 133) replaced the following original item 1 in the definition:

$$\mathbf{codomain}(f) \subseteq \wp(\Theta_{\text{scenario}}(\delta, \gamma)).$$

The remarks below the definition replaced the following original remark:

Note that this function is a “pseudo function” (see remark below Definition 17 (page 20)).

26. On page 53, the first line of the definition of G (Definition 136) replaced the following first line of the original definition of G :

$$G \subseteq \Delta \times \Gamma \times \mathbb{N}^+ \times [0, \infty) \times (\mathbb{N}^+)^3 \times (\wp(\Theta))^{\mathbb{N}^+} \times [1, \infty)^{\Theta \times}$$

27. In Subsubsection 11.4.3, page 54, the two enumerated items were reworded preserving the intended meaning.
28. In Subsubsection 11.5.1.3, page 58, the last two sentences of the original text were revised.
29. In Subsubsection 11.4.5.2, page 57, the symbol ν replaced the symbol $\langle \nu_i \rangle$ in the second sentence of the fourth paragraph.
30. Definition 146 (page 60) replaced the following original definition:

A function $f \in D_0^{D_0}$ is said to *model* $Q \subseteq D_0^2$ if and only if

$$(c, e) \in Q \Rightarrow f(c) = e$$

31. The first line of Definition 148 (page 60) replaced the following first line of the original definition:
 $D: \Delta \times \wp(\Gamma) \times \mathbf{codomain}(\Lambda) \rightarrow \wp(D_0^2) \ni$
32. In Subsubsection 11.5.2, page 59, the first paragraph is a revised version of the original first paragraph, and the third paragraph is a revised version of the original third paragraph.

References

- [1] L. Barbulescu, J.-P. Watson, L. D. Whitley, and A. E. Howe. Scheduling space-ground communications for the Air Force Satellite Control Network. *J. of Scheduling*, 7(1):7–34, 2004.
- [2] K. Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley and Sons, Chichester; New York, 2001.
- [3] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *J. ACM*, 32(3):505–536, 1985.
- [4] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press Series on Computational Intelligence, Wiley-IEEE Press, Hoboken, New Jersey, USA, 2006.
- [5] J. Frank, A. Jónsson, R. Morris, and D. E. Smith. Planning and scheduling for fleets of earth observing satellites. In *Proceedings of the Sixth International Symposium on Artificial Intelligence, Robotics, Automation and Space*, 2001.
- [6] J. H. Friedman. Multivariate adaptive regression splines. *Ann. Statist.*, 19(1):1–67, 1991.

- [7] P. Garza. *Ground Network Users' Guide, Revision 2*. Exploration and Space Communications Projects Division, NASA Goddard Space Flight Center, Greenbelt, Maryland, USA, May 2007.
- [8] R. D. Godfrey. The Communications Link Analysis and Simulation System (CLASS). In *Proc. NASA Johnson Space Center Space Shuttle Tech. Conf., Pt. 2*, pages 787–797, Houston, Texas, USA, Jan. 1985.
- [9] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
- [10] N. Happell, K. L. Moe, and J. Minnix. Scheduling the future NASA Space Network: Experiences with a flexible scheduling prototype. In *Proc., Second International Symposium on Ground Data Systems for Space Mission Operations, SpaceOps 1992*, pages 881–886, Pasadena, California, USA, March 1993. Jet Propulsion Laboratory.
- [11] J. H. Holland. *Hidden Order: How Adaptation Builds Complexity*. The Ulam Lecture Series. Addison-Wesley Publishing Company, Reading, Massachusetts, USA, July 1995.
- [12] W. hu Zhao, J. Zhao, S. hong Zhao, Y. jun Li, and X. Wang. Scheduling algorithm for data relay satellite optical communication based on artificial intelligent optimization. In *Proc. SPIE 8906, International Symposium on Photoelectronic Detection and Imaging 2013: Laser Communication Technologies and Systems*. SPIE, August 2013.
- [13] I.-J. Jeong, G. Papavassilopoulos, and D. Bayard. Task scheduling on spacecraft by hybrid genetic algorithms. *Proc. 1999 IEEE International Conference on Robotics and Automation, Detroit, Michigan, USA*, 1:441–446, 1999.
- [14] K. A. D. Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, Cambridge, Massachusetts, USA, 2006.
- [15] T. Kaplan, J. Freedman, D. Wampler, A. Musliner, and C. Ruseau. CLASS Interference Analysis System. *National Telesystems Conference, NTC-92*, pages 12/1–12/7, May 1992.
- [16] V. Kecman. High dimensional function approximation (regression, hypersurface fitting) by an active set least squares learning algorithm. School of Engineering Report 643, The University of Auckland, Auckland, NZ, 2006.
- [17] C. Lim and E. Sim. Production planning in manufacturing/remanufacturing environment using genetic algorithm. In *GECCO '05: Proceedings of the 2005 conference on genetic and evolutionary computation*, pages 2217–2218, New York, NY, USA, 2005. ACM.
- [18] R. M. Mickey, O. J. Dunn, and V. A. Clark. *Applied Statistics: Analysis of Variance and Regression*. Wiley Series in Probability and Statistics. Wiley-Interscience, 3rd edition, January 2004.
- [19] Mission Services Program Office, Goddard Space Flight Center, National Aeronautics and Space Administration, Greenbelt, MD, USA. *Space Network Users' Guide (SNUG) (450-SNUG (Revision 9))*, August 2007.
- [20] NASA Goddard Space Flight Center, Greenbelt, Maryland, USA. *CLASS ACRS/TLAS Operator's Manual and Reference (NCC 98) (Draft)*, 1998.
- [21] E. Novak and H. Woźniakowski. *Tractability of Multivariate Problems: Linear information*, volume 1. Tracts in Mathematics 6, European Mathematical Society, Zurich, Switzerland, 2008.
- [22] J. B. Odubiyi and D. R. Zoch. A heuristic approach to incremental and reactive scheduling. In *Proc. Goddard Conference on Space Applications of Artificial Intelligence*, pages 127–140, Greenbelt, Maryland, USA, 1989. NASA Goddard Space Flight Center.
- [23] K. L. Priddy. *Artificial Neural Networks: An Introduction*. SPIE Press, 2005.
- [24] H. Rabitz and Ö. F. Alis. General foundations of high-dimensional model representations. *Journal of Mathematical Chemistry*, 25(2–3):197–233, 1999.
- [25] J. L. Rash. Heuristic evolutionary search in space communications scheduling. In G. A. Soffen, editor, *Research and Technology Report*, pages 2–4. NASA Goddard Space Flight Center, Greenbelt, Maryland, 1995.
- [26] J. A. Rumerman. *NASA Historical Data Book*, volume VI (NASA Space Applications, Aeronautics and Space Research and Technology, Tracking and Data Acquisition/Support Operations, Commercial Programs, and Resources, 1979-1988) of *NASA Historical Data Book, NASA SP-2000-4012*. National Aeronautics and Space Administration, Washington, DC, USA, 2000.
- [27] T. P. Ryan. *Modern Regression Methods*. Wiley-Interscience Series. John Wiley and Sons, New York, NY, USA, 2008.
- [28] G. Syswerda. Schedule optimization using genetic algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms*, chapter 21. Van Nostrand Reinhold, New York, New York, USA, 1991.
- [29] G. Syswerda and J. Palmucci. The application of genetic algorithms to resource scheduling. In R. K. Belew and L. B. Booker, editors, *ICGA*, pages 502–508. Morgan Kaufmann, July 1991.
- [30] M. D. Vose. *The simple genetic algorithm: foundations and theory*. MIT Press, 1999.
- [31] L. W. Wajda. *Network Planning and Analysis System (NPAS) User's Guide (Fundamentals)*. Vol. 1 (TDAC-NUG-SAS-001), NASA Goddard Space Flight Center, Mission Operations and Data Systems Directorate, rev. 1 edition, July 1997.
- [32] M. B. Wall. *A Genetic Algorithm for Resource-Constrained Scheduling*. PhD thesis, Massachusetts Institute of Technology, Boston, MA, USA, June 1996. Supervisors Mark Jakiela and Woodie C. Flowers.
- [33] Y. F. Wong and J. L. Rash. A method for interference mitigation in space communications scheduling. In *IEEE Military Communications Conference (Milcom 91)*, volume 2 of 3, pages 431–435, Piscataway, New Jersey, 4–7 November 1991. IEEE Communications Society, IEEE.

Index of Symbols and Selected Terms

- $<$, 20, *see* interval, ordering relation, *see* interval, sets of intervals, ordering relation
 A^* , 21, *see* antenna
 A_0 , 21, *see* antenna
 A_r , 60, *see* regression analysis
 C , 23, *see* prototype communications event
 C_0^{PRM} , 26
 D , 60, *see* training data
 D_0 , 50
 $F_{\text{crossover}}$, 52
 F_{fitness} , 51, *see* fitness, function
 F_{mutation} , 52
 $F_{\text{randomselection}}$, 51
 F_{META} , 55
 $F_{\text{new-g}}$, 55
 G , 53, *see* Type-G, problem
 G_{scenario} , 53
 I , 22, *see* potential interference interval
 L , 22, *see* communications link
 L^* , 21, *see* communications link
 L_0 , 22, *see* communications link
 M , 22, *see* mission event
 M^* , 21, *see* mission event type
 M_{skips} , 24, *see* mission event
 M_{type} , 24, *see* mission event type
 M_T , 24, *see* mission event
 O , 21, *see* outage interval
 P , 21, *see* POCC operation period, *see* POCC
 P_{max} , 25, *see* POCC operation period
 R_0 , 23, *see* user requirement
 S_0 , 21, *see* Ground Network, *see* Space Network, *see* station
 S_0^{SA} , 21
 U , 24, *see* user
 U_0 , 21, *see* user
 V , 22, *see* communications view period
 W_{steps} , 53
 X^Y , 19, *see* function
 Y , 23, *see* service
 Y^V , 24, *see* communications view period, *see* service instantiation
 Y^I , 24, *see* service instantiation
 Y_{max}^I , 25
 $[\bullet, \bullet]$, 19, *see* interval
 Δ , 50, *see* problem domain of Type G, *see* Type G, problem domain
 Δ_{dim} , 50
 Γ , 38, 50, *see* Type-G, problem scenario
 Γ_{dim} , 50
 Λ , 40, 59, *see* characterization function
 \emptyset , 19, *see* empty set
 Ω , 19, *see* universe of discourse
 Ω_{F} , 19, *see* set of all finite sets
 Φ , 23, *see* priority
 Φ_0 , 22, *see* priority
 Θ , 26, *see* schedule, 50, *see* Type-G, problem scenario
 Θ_{RND} , 35
 Θ_{scenario} , 50
 Υ , 60
 Ξ , 20, *see* sequence, of all members of a finite set without repeats
 Ξ^{**} , 20, *see* sequence, of some members of a set allowing repeats
 Ξ^* , 20, *see* sequence, of some members of a set without repeats
 \setminus , 19, *see* set, difference
 \bullet , 19
 \circ , 19, *see* function, composition
 κ^{GN} , 29
 κ^{SN} , 28
 $\langle \bullet \rangle$, 20, *see* sequence, *see* tuple
 \mathbb{N} , 19
 \mathbb{N}^+ , 19
 \mathbb{R} , 19, *see* real numbers, set of all
 \mathbb{Z} , 19
 \mathbb{Z} , 19, *see* interval, set of all integer intervals
 ψ , 38
J, 31
MAXALLOWEDRTNRATE, 22
RND, 21, *see* random, subset of finite set, *see* set, random subset
S, 21, *see* string
chngsvcant, 33
chngsvcdur, 32
chngsvcesta, 32
codomain, 19, *see* function, codomain
cutexcesspei, 33
dom, 19, *see* function, domain
endpts, 28
endpts_{seq}, 28
end^P, 27
fitness, 32
fitness[#], 38
interf^{*}, 28
len, 20
maxsepsat^{P*}, 27
max, 21, *see* set, maximum value of
minsepsat^{P*}, 27

min, 21, *see* set, minimum value of
modsvc, 32
replacepi, 33
resourceusage, 28
rndint, 20, *see* random, integer
rndmember, 20, *see* set, random member
rndpeis, 35
rndpei, 32
rndsvcs, 34
rndsvc, 32
rtndatarate^{COMBINED}, 31
satisfied^{*}, 31, *see* user requirement, satisfaction
satisfied^{PEI}, 31
satisfied^R, 31
schedolpairs, 27
skipsat^{R*}, 26
skip^{FILL-R*}, 26
slipsvc, 32
start^P, 27
stnsw^{PEI}, 30
swapearlypeionr, 34
swapmidpeionr, 34
swappeionr, 33
swappeionu, 34
swappei, 33
tref, 24, *see* mission event
usage^{STATION-SA-ENDPTS}, 30
violations^{GN-ENDPTS}, 29
violations^{GN}, 30
violations^{MAXSEP*}, 27
violations^{MINSEP*}, 27
violations^{RTNRATE}, 31
violations^{SA-ENDPTS}, 30
violations^{SA}, 30
violations^{SKIP*}, 26
violations^{SKIPFILL*}, 27
violations^{SN-ENDPTS}, 29
violations^{SN}, 29
violations^{STATION-SA-ENDPTS}, 30
violations^{STNSW}, 30
violations^{GN-ENDPTS}_{B,C,D}, 29
violations^{SN-ENDPTS}_{B,C,D}, 29
violation^{RTNRATE}, 31
 $|\bullet|$, 19, *see* cardinality
 \wp , 19, *see* power set
 $+$, 20, *see* interval, endpoint
 $-$, 20, *see* interval, endpoint
 a_{regr} , 60
 $f_{\text{estimation}}$, 59–61, 65
 $f_{\text{g-execute}}$, 54
 \times , *see* Cartesian product

algorithm, 8, 35, 38, 61
 Algorithm 3: Scheduling Problem Optimal-Internal-Parameters Estimation, 41
 Algorithm 2: S[#] Problem, 38
 Algorithm 4: Type-G Problem, 53
 Algorithm 5: G[#] Problem, 61
 Algorithm 1: Optimal-Schedule Generation, 35
 antenna, 5, 10–13, 16–18, 21, *see* A^* , *see* A_0
 cardinality, 19, *see* $|\bullet|$
 Cartesian product, 19, 20
 characterization function, 59, *see* Λ
 CLASS, 8, *see* Communications Link Analysis and Simulation System (CLASS), 17, 22
 codomain, 19, *see* function
 communications event window, 23, 24
 communications link, 21, *see* L^* , 22, *see* L , *see* L_0
 Communications Link Analysis and Simulation System, 8, 17, 22
 communications view period, 16, 17, 22, *see* V , *see* Y^V
 computer, 8, 13, 15, 42, 43
 quantum, 13
 run time, 38, 53, 56, 57, 62
 constraints, 5, 7, 8, 11–13, 29, 43
 GN Forward and Return, 28, *see* κ^{GN}
 SN Forward and Return, 28, *see* κ^{SN}
 Space Network resource usage, 28, *see* κ^{SN}
 crossover, 13, 32, 52, *see* $F_{\text{crossover}}$
 domain, 19, *see* function
 empty set, 19, *see* \emptyset
 evolutionary algorithms, 13
 evolutionary search, 8, *see* genetic algorithm, *see* probabilistic search, 9, 13–15, 39, 49–51, 54, 55, 57, 59
 fitness, 32, 37, 38
 function, 32, 51, *see* F_{fitness}
 S[#] problem, 38, *see* **fitness**[#]
 Type-G problem, 50
 function, 19, 20, *see* X^Y
 codomain, 19, *see* **codomain**
 composition, 19, *see* \circ , 55
 domain, 19, *see* **dom**
 G[#], 49, *see* Type-G, meta problem, 54
 G[#] problem, 56
 optimization, 56
 G[#], 49, *see* Type-G, meta problem, 54
 G[#] problem, 61
 algorithm, 61
 regression, 61
 G-algorithm, 53, 56

genetic algorithm, 8, *see* evolutionary search, 13–15, 37, 50, 54
 internal parameters, 15
 Ground Network, 5, 8, 21

hypersurface, 15, 40, 42, 60, 65
 fitting, 55

index, 20
 sequence, 20
 tuple, 20

interference, 5, 7, 8, 10, 13, 17, 18, 22, 23, 42, 43, 65
 interference instance, 28, *see* **interf**^{*}
 internal parameters, 9, 15, *see* genetic algorithm, internal parameters, 35, 37–43, 46, 49, 54–57, 59, 61, 64, 65
 estimation, 40, *see* $S^{\#\#}$ Problem, 58, *see* $G^{\#\#}$ Problem
 estimation function, 9, 40, 41, 44, 55, 58, 59, 65
 optimization, 15, *see* $G^{\#}$, 37, *see* $S^{\#}$

interval, 19, *see* $[\bullet, \bullet]$, 20
 endpoint
 +, 20
 -, 20
 ordering relation, 20, *see* <
 set of all integer intervals, 19
 sets of intervals
 ordering relation, 20, *see* <

law of diminishing returns, 46

mapping, 19, *see* function
 meta problem, 54
 meta-problem scenario, 55, *see* **meta**, 56, *see* Type-G, meta problem, 58
 method, 8
 mission event, 22, *see* M , *see* **tréf**, *see* M_{skips} , *see* M_T
 mission event type, 21, *see* M^* , 24, *see* M_{type} , 26
 model, 9, 40, 47–49, *see* regression analysis, 59, 60, 65
 mutation, 13, 32, 52, *see* F_{mutation}

optimal, 8–10, 12–16, 35, 37, 38, 40–44, 46, 49
 optimization, 8, 9, 13–15, 37–44
 recursive, 56, *see* recursion, 57, *see* recursion
 optimum, 13, 37, 39, 57
 ordered pair, 20, *see* tuple
 outage interval, 18, 21, *see* O

POCC, 17, 18
 POCC operation period, 21, *see* P , 25, *see* P_{max}
 potential interference interval, 18, 22, *see* I
 power set, 19, *see* \wp
 priority, 18, 22, *see* Φ_0 , *see* Φ
 probabilistic search, 13–16, 37, *see* evolutionary search, *see* genetic algorithm, 55

problem domain of Type G, 50, *see* Δ
 problem scenario, 50, *see* Γ
 characterization function, 59, *see* Λ
 dimension, 50, *see* Γ_{dim}
 meta, 54, 58
 Type-G, *see* Γ , 58

process, 8
 process models, 43
 prototype event, 23, *see* C , 66
 prototype-event instantiation, 26, *see* C_0^{PRM}
 pseudo function, 20, 51–53

random, 14
 integer, 20, *see* **rndint**
 member of a finite set, 20, *see* **rndmember**
 number, 14, 16
 selection, 51
 subset of finite set, 21, *see* **RND**

real numbers, set of all, 19, *see* \mathbb{R}
 recursion, 56, 57, 61
 regression analysis, 9, 13, 40–42, 59, 60, *see* A_r , 61, 63–65
 requirement, 22, *see* user requirement
 run time, 8, *see* computer, run time, 38, 46, 48

$S^{\#\#}$, 38, 40
 problem, 37
 $S^{\#}$, 15, *see* internal parameters, optimization, 37–39
 schedule, 5, 7, 18, 26, *see* Θ , 65
 optimized, 43
 scheduler, 7, 8, 13, 18, 42, 43
 optimizing, 7–9, 43, 44
 scheduling
 objective, 26
 system, 7, 9, 13, 16, 17, 41–43
 sequence, 20, *see also* tuple
 elements of, 20, *see* $\langle \bullet \rangle$
 index, 20
 length, 20, *see* **len**
 of all members of a finite set without repeats, 20, *see* Ξ
 of some members of a set allowing repeats, 20, *see* Ξ^{**}
 of some members of a set without repeats, 20, *see* Ξ^*
 subsequence, 20

service, 23, *see* Y
 service instantiation, 24, *see* Y^V , *see* Y^I
 set, 19
 difference, 19, *see* \setminus
 maximum value of, 21, *see* **max**
 minimum value of, 21, *see* **min**
 random member, 20, *see* **rndmember**
 random subset, 21, *see* **RND**
 specification notation, 19
 set of all finite sets, 19, *see* Ω_F

- skip factor, 23–26
 Space Network, 5, 8, 21
 Space Network Users' Guide, 28
 station, 21, *see* Ground Network, *see* Space Network
 string, 21, *see* **S**
- test data, 41, *see* training data, 65
 training data, 41, 60, *see* D , *see* $G^{\#}$, *see* regression analysis, 63
- tuple, 20, *see also* sequence, *see* $\langle \bullet \rangle$
- Type-G, 50, 53, 64
 - meta-problem, 49, *see* $G^{\#}$, *see* $G^{\#\#}$, 54–59, 63
 - chain, 56, 57
 - problem, 53, *see* G , 58
 - problem domain, 50, *see* Δ , 59
 - problem scenario, 50, *see* Γ , 58, 64
- universe of discourse, 19, *see* Ω
- user, 5, 7, 8, 10–12, 18, *see* U_0 , *see* U
- user requirement, 14, 17, 18, 22, 23, *see* R_0 , 24
 - generic, 22
 - satisfaction, 16, 31, *see* **satisfied***
 - specific, 22

