Gemini Automated Scheduler Trade Study

By Sebastian Raaphorst, Bryan Miller, Arturo Nuñez, Sergio Troncoso, Kristin Chiboucas, and Fredrik Rantakyro.

April, 2022

Table of Contents

1 Introduction

For several years, Gemini has been planning to implement an automated Scheduler to produce plans of observations across both sites. The current system of Queue Coordinators manually creating queues for every possible set of conditions is labor intensive and does not adapt well to a variety of situations, such as nighttime faults and ToOs (targets of opportunity). With the upcoming deployment of the Vera C. Rubin Observatory and the launch of the 10-year LSST (Legacy Survey of Space and Time) project, automated scheduling becomes more important, and the NSF has provided Gemini with the funds for the GEMMA (Gemini in the Era of Multi-Messenger Astronomy) project.

The primary motivator for automated scheduling is to handle the large increase in the number of ToOs, but will also decrease the required staff effort for scheduling and maintain or improve completion rates for remaining all programs. The Scheduler will leverage the Gemini Program Platform (GPP) and is a key part of the OCS Upgrades Program.

In order to assess the most flexible, advantageous technique for automated scheduling that satisfies the requirements laid out by the GEMMA program, we have spent the last several months performing a trade study, consisting of a detailed literature review of the techniques used by other observatories, and prototyping a number of such techniques to determine their performance and ability to meet our needs.

In this report, we give a brief description of the techniques prototyped, our findings, and a Pugh table to illustrate which techniques of those studied are the most promising moving forward.

2 Techniques Considered

The general idea used in all of the following algorithms is that, for each observation, we combine various relevant factors such as ranking band, program completion, target visibility, timing window, hour angle, and conditions matching, which is used to calculate the observation's score.

After a detailed review of the literature, we tested ten algorithms that can be subgrouped into five families of algorithms to examine and prototype, namely:

1. ILP (Integer Linear Programming) modeling, comprising Gurobi and CBC.
2. Genetic algorithms and Multi-Objective Evolutionary Algorithms.
3. Simple optimization heuristics (hill-climbing, simulated annealing, great deluge, and brute force); and
4. A greedy approach; and
5. The Spike package.

We will explain each technique and our motivation for our selections below.

2.1 Integer Linear Programming

Integer Linear Programs (ILPs) are widely used to solve maximization / minimization techniques. The fundamental notion is to model the problem to be solved as a set of variables that can take on integer values and then formulate a set of constraints using the variables to represent the constraints of the problem under consideration. Lastly, an objective function is provided, either to be maximized or minimized.

Las Cumbres Observatory (LCO) - consisting of 23 telescopes at seven sites across the world - uses ILPs in their automated scheduling system, which has been running for several years and shown to be robust and effective for their needs. (Lampoudi et al, 2015)

Many optimization problems restrict the variables to binary values. In the case of LCO, the night is divided into a collection of time slots $T$of a fixed length, and a set of observations O- each with an associated priority - is assigned to the time slots in order to maximize the total sum of the priorities of the observations.

A variable is created for each observation $o \in O$ (with priority o_p) and time slot t for which the observation can be scheduled, i.e. y_{ot}, and constraints are added to ensure that:

1. Each observation is scheduled no more than once.
2. Each time slot is occupied by at most one observation.

The objective function to maximize then becomes:

$$\sum_{o \in O} \sum_{t \in T} o_p y_{ot}$$

ILPs are a well-studied approach to solving optimization problems, such as the traveling salesman problem and scheduling problems (more generally, the tool shop job problem). Additional requirements can be modeled as constraints, and a variety of both commercial (e.g. Gurobi, ILOG CPLEX, Mosek) and free, open-source (IBM's COIN-OR CBC, GNU's glpk) products are available. The answer returned by ILPs always meet the constraints and maximize (or minimize) the objective function, and are thus optimal. Given that the type of problems typically solved by ILPs are NP-complete, a class of problems which includes scheduling problems, the time required to solve these problems scales up rapidly with the number of variables and constraints. Commercial solvers allow a "margin of inoptimality" (i.e. a certain distance away from optimality) in order to exercise some level of control over the number of computations if the user is willing to accept a "good enough" solution.

Some commercial products offer "multi-objective function" problem solving,, i.e. they allow the user to supply multiple objective functions to maximize or minimize. This, however, is typically accomplished using a blending of the objective functions given weighting, or the objective functions are solved sequentially.

One advantage of ILPs is that the same algorithm can be used to schedule on any time scale: short-term / real-time, mid-term, and long-term.

After discussions with Las Cumbres Observatory, it was decided that while both the CBC and the Gurobi algorithms might both be feasible representative candidates for this family of algorithms, LCO had been using Gurobi successfully for many years. Gurobi was originally used because of its speed, but LCO is currently conducting experiments with CBC as well to determine the differences in speed and if it is significant.

Additionally, Gurobi allows for models with multiple objective functions, whereas the open source libraries only allow for single objective functions and the user must handle the blending of the multiple objective functions, which is not a trivial task. This allow us to maximize and minimize multiple objective functions simultaneously and as a result, allow for more flexibility and satisfaction of Gemini requirements: we thus, for the purposes of this trade study, selected the Gurobi ILP solver as the solver to represent the ILP family of algorithms due to its speed, flexibility, and trial licenses.

2.2 Genetic Algorithms

Genetic algorithms are a long-used optimization evolutionary algorithm modeled after natural selection: given a fitness function, a population of solutions (referred to as *chromosomes*, which are a sequence of *genes*) is instantiated, either deterministically or stochastically, and then a number of iterations is performed where the following operations are executed on the population:

1. *Crossover:* two chromosomes are chosen from the population, and *reproduce* to create two offspring, usually each with half of their genes coming randomly from each parent. Of the original two chromosomes and their two offspring, the two most fit are included in the next generation of solutions. (e.g. P1=ABCD, P2=EFGH, C1=ABGH, C2=EFCD)
2. *Mutation:* a chromosome is chosen from the population, and one or more of its genes is randomly mutated. If the new chromosome increases in fitness, it replaces the original chromosome in the next generation of solutions.

Other custom operations can be added as appropriate.

This proceeds until a defined stopping criterion is achieved, which usually consists of:

1. A fixed number of iterations (generations) is reached.
2. A fixed number of iterations where no improvement in the highest fitness is seen.
3. A certain amount of time has passed.

The solution with the highest fitness is maintained throughout the process to be presented as the final solution.

We model our genetic algorithm Scheduler prototype after one proposed by CFHT (the Canada-France-Hawaii Telescope), as per Mahoney et al, 2012. Their approach is slightly unique:

1. It builds the chromosomes (candidate plans) in such a way that every gene (observation) that is viable (can be observed during the time period represented) is included in one chromosome.
2. A chromosome can consist of “empty space”, and is considered optimal if it is at least filled to 85% capacity.
3. When selecting two chromosomes for crossover, one is chosen at random from the top 25% of the population, and the other is chosen at random from the whole population.
4. In addition to crossover, an operation called *interleaving* is included, where the genes of the two chromosomes are alternated back and forth to produce two children. (e.g. P1 and P2 as above yield C1=AEBF, C2=EAFB)
5. Mutations consist of interchanging the positions of two genes in a chromosome, dropping genes from a chromosome, or adding genes to a chromosome.

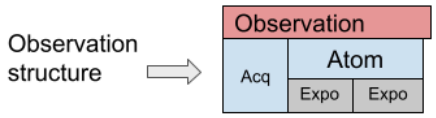
Due to Gemini having two telescopes with overlapping plans, the problem becomes more difficult to maintain as we must ensure that the plans that result from the operators are valid.

We have two pools of chromosomes: one for each site (Gemini North and Gemini South), and use the following operators to modify the chromosomes in a population to produce the next generation for each pool:

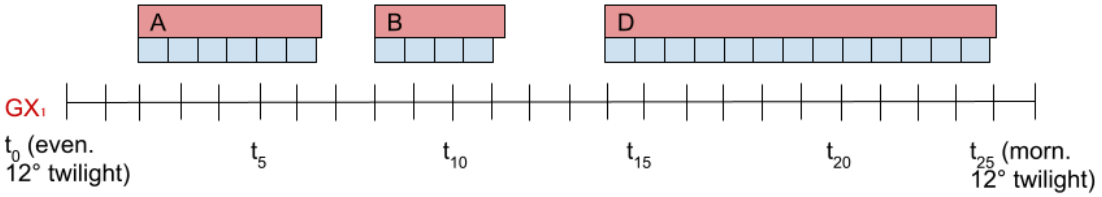
1. Partial insertion: Insert a part of an observation into a chromosome such that it abuts either the beginning, end, or another observation in the chromosome.
2. Mate: We select two chromosomes from the same pool and select a crosspoint for each chromosome. We then mate the chromosomes in such a way that we swap the data after the crosspoints, leaving out any observations that would interfere with the observations before the crosspoints.
3. Interleave: We select two chromosomes from the same pool and swap an observation from one chromosome with one observation from the other chromosome such that this does not cause observations to overlap.
4. Mix: We swap out one observation at random for another at the same site, provided the new observation does not interfere with the remaining existing observations in the chromosome.
5. Partial insert expand: This is a subcase of mate, where when a partial observation from the first chromosome is inserted into the second chromosome, it is placed in such a way that there is a gap, thus providing room for the partial observation to expand. Note that in the diagram linked to below, for this example, the green bars above the chromosome represent the time slots where the partial observation can be scheduled.

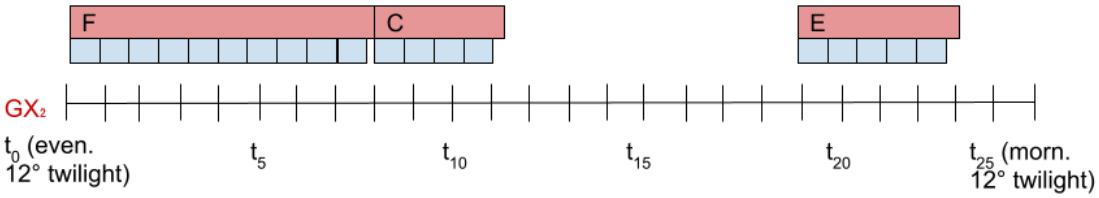
A visual depiction of the operators are shown in the following figures:

Initial Structure

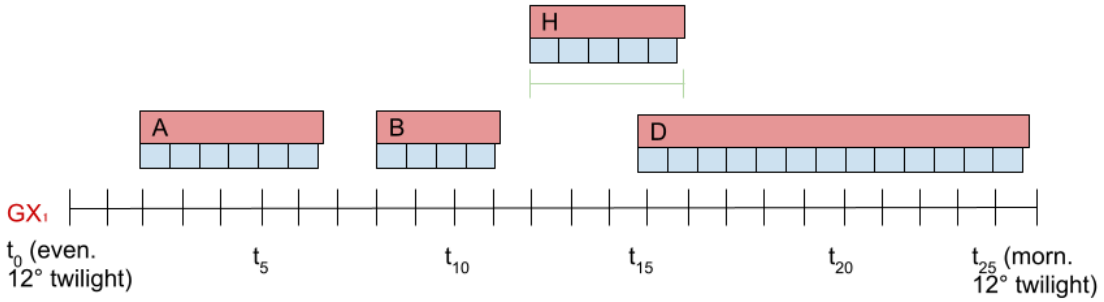


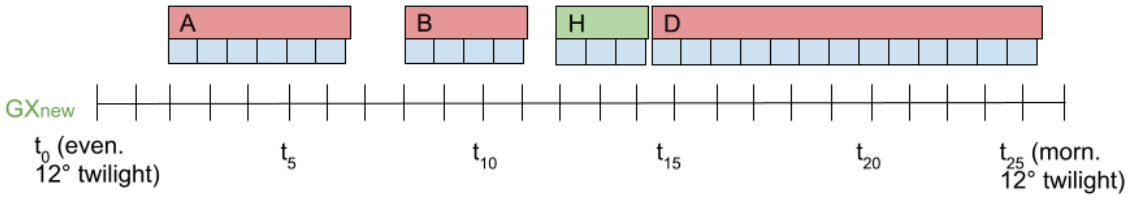
Chromosome structure (by site)



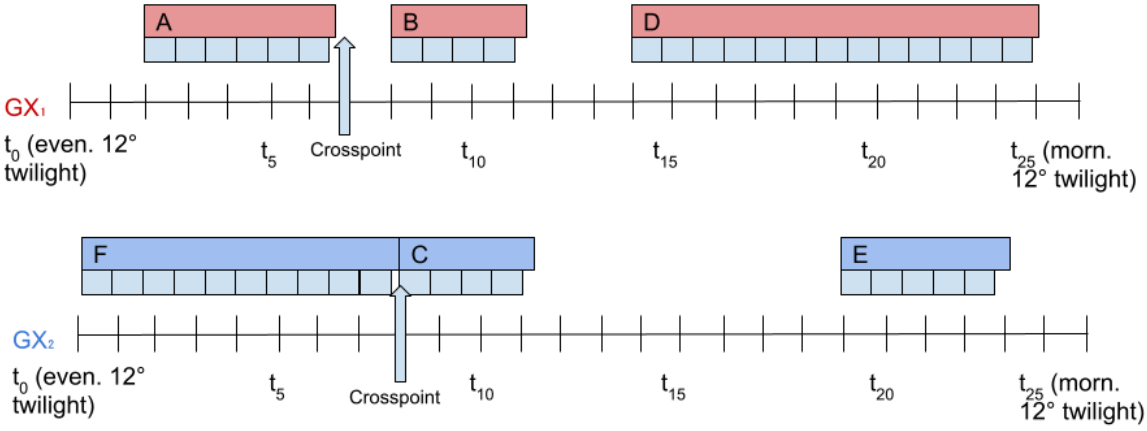


Partial Insert



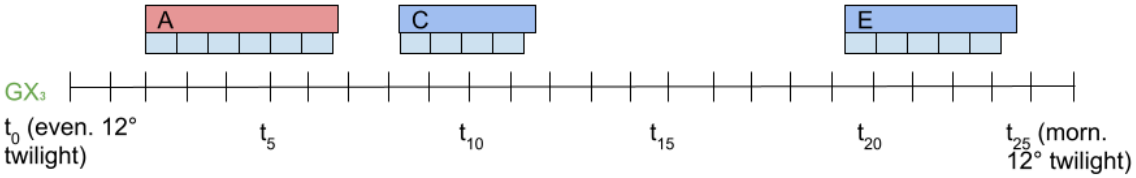


Mate

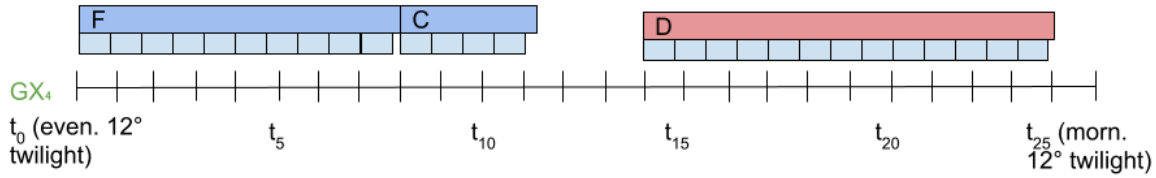


Mate

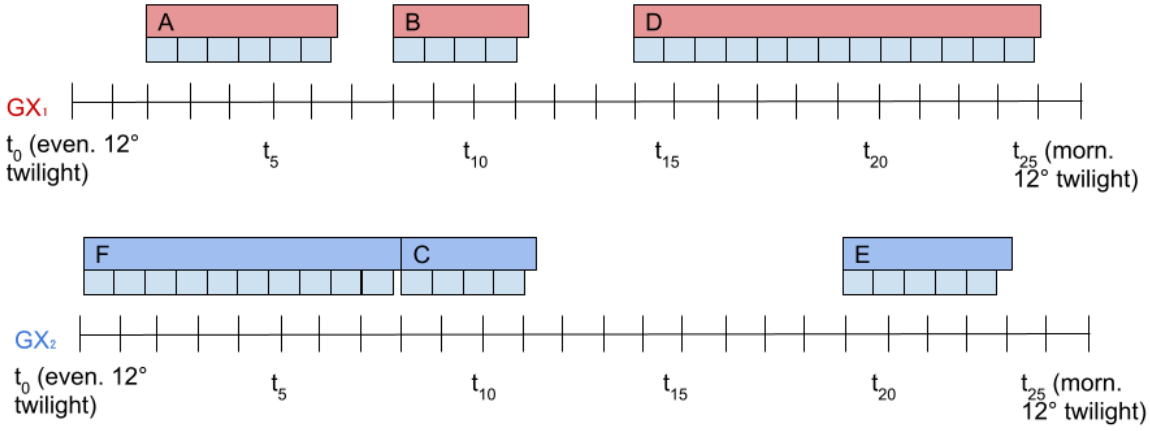
Crosspoint: t_7



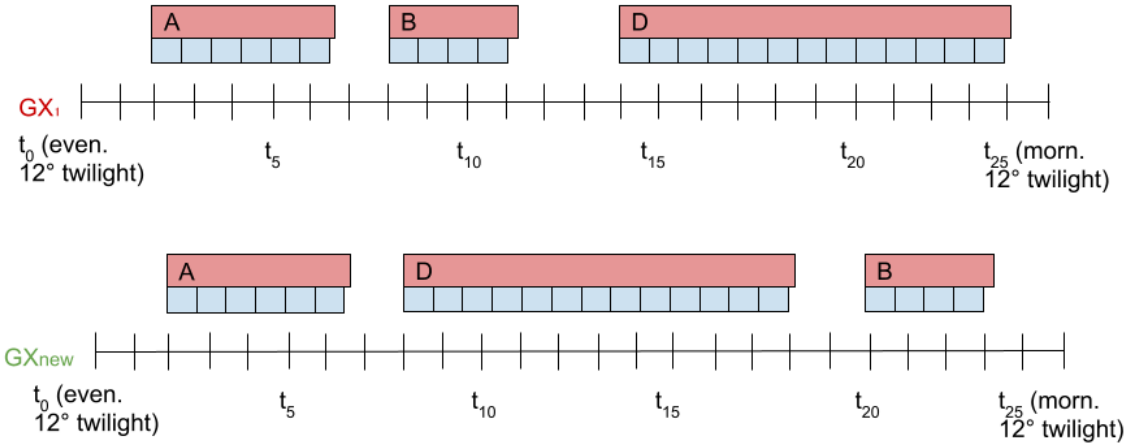
Crosspoint: t_{11}



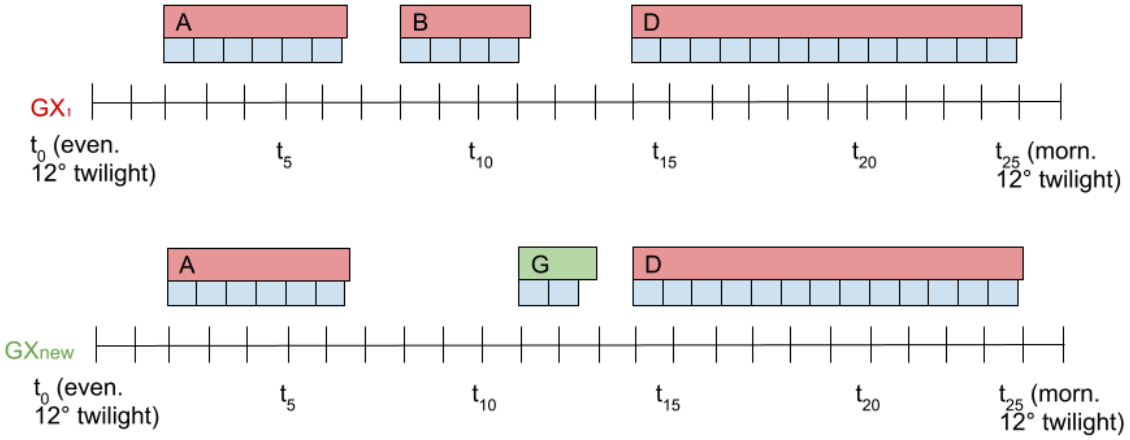
Interleave



Swap

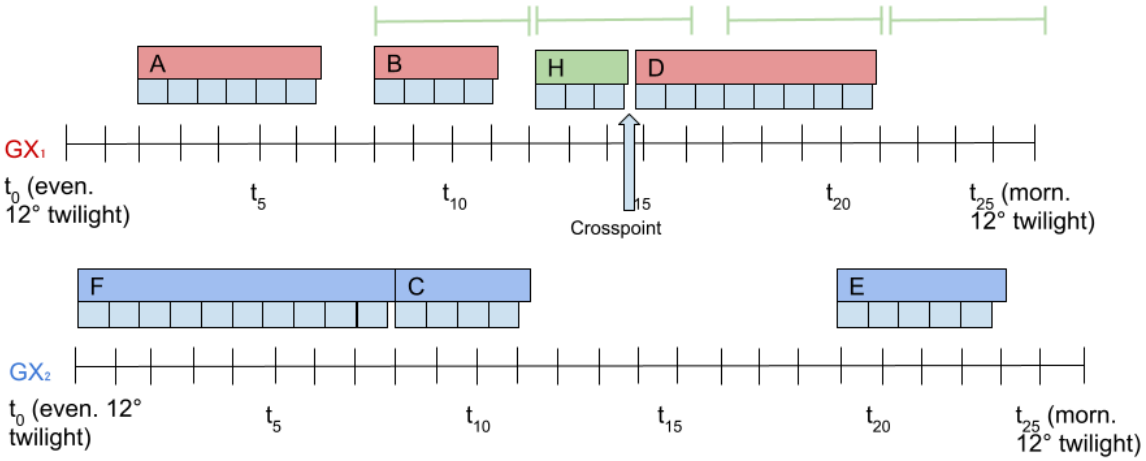


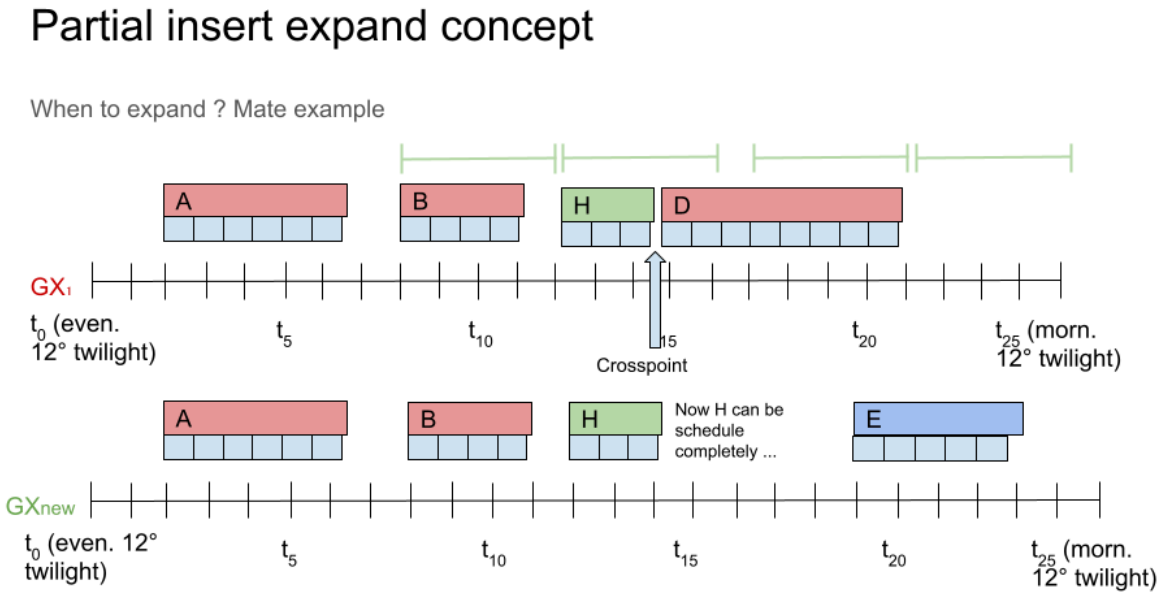
Mix



Partial insert expand concept

When to expand ? Mate example





Note that in the above diagram examples, red and blue observations represent observations that can be fully scheduled, and green observations represent observations that have only been partially scheduled.

Genetic algorithms can be used to solve a wide range of problems and typically yield very good results when tuned properly. A well-known example is the design of the 2006 NASA ST5 spacecraft antenna. (https://en.wikipedia.org/wiki/Genetic_algorithm)

One feature of genetic algorithms is that due to the randomness involved in the operations on the chromosomes, each run can (and usually will) generate a different solution with variable fitness: thus, the algorithm can be run multiple times with the same input and the best solution chosen. On a multi-core system, multiple instances can be simultaneously run.

An advantage of genetic algorithms is that they can be generalized to have multiple fitness functions to be maximized / minimized, in which case, they are called *multi-objective evolutionary algorithms* (MOEAs). A solution is considered optimal in these algorithms if changing the representative chromosome decreases the fitness with respect to one of the objective functions.

We attempted to use the open source RTS2 (Remote Telescope System 2, Kubanek, 2008) - a telescope scheduling specific MOEA - which has been used in production by several telescopes - but we were unable to compile the code, possibly due to its age. As MOEAs can be quite difficult to code, it is worth noting that there are several open-source, well-maintained MOEA frameworks available, such as MOEA framework at http://moeaframework.org. The requirements we have examined so far should be satisfiable using genetic algorithms and we have focused on pure genetic algorithms in the study (in particular, the implementation of

genetic algorithms proposed by CFHT, 2012). The Pugh table below is limited to genetic algorithms with notes where MOEAs might potentially offer us an advantage. Further study would be required to determine if this power is needed.

Disadvantages to the genetic algorithm approach with regards to the Gemini Scheduler requirements are primarily that scheduling AND / OR groups is likely to be possible in the creation of the initial population, but when executing operators on the chromosomes, becomes difficult or impossible to maintain. Additionally, due to the heuristic nature of the algorithms and the number of suboptimal peaks in the search space, it is possible that the score we obtain between two runs of this Scheduler might differ vastly, and are non-reproducible given the same data.

2.3 Simple Metaheuristics (Hill-Climbing Algorithms aka HCAs)

Due to their simplicity, we also implemented a family of simple optimizers based on modifications of hill-climbing. The general idea in hill-climbing is that the solution space is represented as a multi-dimensional landscape, where each point represents a candidate solution and has a defined neighborhood in the landscape. The algorithm begins by randomly selecting a point in the landscape, and then proceeds to examine its neighborhood: if there is a member of the neighborhood with a better than or equal fitness, it moves to the neighbor with the best fitness. This ensures that the algorithm never decreases its fitness. The algorithm terminates when it either has no neighbors that are at least as fit as the current solution, or the fitness has not increased for a specified number of iterations.

The name represents the imagery of climbing a hill to reach the highest point; however, hill-climbing algorithms perform poorly with respect to certain problems if there are many local maxima: once a local maxima is reached, the algorithm must terminate, and local maxima may be substantially less fit that the global maximum.

Many modifications of hill-climbing have been derived to help overcome this problem. Amongst them, the ones we prototyped include:

1. *Simulated annealing:* This variant is based on the properties of the cooling of metal. At each stage, the algorithm is allowed to move to a less fit solution with a certain probability. This probability decreases after each iteration. Thus, local maxima can be escaped and more of the landscape can be explored. The disadvantage to simulated annealing is that it has a large number of parameters that must be tuned for the algorithm to be effective.
2. *Great deluge:* (Dueck 1993) This algorithm is based on rainfall. The landscape starts off “dry.” With each move, a given, fixed amount of rainfall occurs. At each iteration, the algorithm can move to any neighbor, provided that the neighbor is above the current level of rain. As in simulated annealing, this allows us to move to worse solutions to escape local maxima. It has been shown to be as effective as simulated annealing, and easier to tune due to the small number of parameters.

In addition to these three algorithms, we implemented a *brute force* approach, which, given a set of observations and a time length in which to schedule them, examines every possible solution and produces the candidate guaranteed to be optimal. Since scheduling is NP-complete, this technique only works for small numbers of observations and short time frames.

In general, these algorithms do not scale well to scheduling anything other than one night, and furthermore, many of the requirements for the Scheduler are difficult or perhaps even impossible to capture by the algorithms. Again, as they are heuristics, given a night, the plan produced is likely not reproducible. In the Pugh table below, given their relative computational equivalency, we focus on HCA - or Hill Climbing Algorithms - and ignore the brute force approach as it is unfeasible for all but the smallest data sets. The HCAs are all generally comparable in terms of quality, with simulated annealing and great deluge simply more likely to reach better solutions. Due to the ease to tune the great deluge algorithm, this is the focus of the HCA column in the Pugh table below.

2.4 Spike

Spike (Johnston and Miller, 1993) is an automated scheduling package that was specifically designed to be used in the Hubble project in 1990. While the original system was written using a pseudo-neural network, it has been altered over the years and now is a constraint satisfier. It offers a number of different scheduling techniques and support for ground-based telescopes. It is currently maintained by Mark Guiliano.

Spike is written in Common Lisp, and thus does not fit well into the Gemini ecosystem of software. While the code is available, there are no software engineers on the team that are competent Common Lisp programmers, and many of the modern technologies that are in use in the new OCS (e.g. GraphQL) are not available for Common Lisp.

Spike offers a set of five algorithms, but only one of these algorithms takes priority of observations into account, which is a strict requirement for Gemini. It is thus this variant that we focus on in the Pugh table below, as we realized in our earlier experiments that the other four would not be sufficient for our needs.

Whether Spike is currently being used for automated scheduling at any ground-based telescopes is unclear, as most of the data regarding Spike usage is outdated.

2.5 GreedyMax

GreedyMax was written and is currently maintained by Bryan Miller at Gemini as a proof-of-concept automated Scheduler. Matt Bonnyman translated the original code into Python, and Sergio Troncoso has refactored the code for organizational purposes and to simplify understanding and maintaining it in the future.

The algorithm attempts to mimic how a human queue coordinator makes a plan. It evaluates the scores for all valid observations for the current conditions in the unscheduled time slots and selects the observation with the maximum score. The observation is placed in the plan based on the integral of the score function over either the total time needed for the observation or the available schedule window, whichever is shorter. Other aspects of the algorithm are:

- It schedules as many long observations as it can, splitting them where needed. Therefore, it avoids the need for fixed-length observing blocks. It can break observations at sequence atoms once those are defined.
- There is a minimum visit length (e.g. 60 min) In order to avoid a lot of short visits. If an observation is shorter than the minimum, then the whole observation is scheduled. The minimum visit length could be dependent on instrument and mode.
- If a scheduling block is within the minimum visit length of the edge of the available interval then it attempts to push the observation to the boundary to avoid small gaps.
- If a new program or observation is started then it increases the scores, via the updated metric, for the remainder of the observation (if any) and the other observations in the program.
- Time for calibrations and additional acquisitions are added as needed, and the calibrations are added in the correct order.

The process is repeated for each unscheduled time interval until all the intervals are filled or determined to be unschedulable. It typically fills over 95% of a night. Besides being fast, taking about 10 seconds to schedule two telescopes together, the algorithm is very flexible since actions can be taken after each observation is scheduled. Therefore, calibrations and other associated observations, slew times, etc. can be factored in as the plan is built up.

While designed to schedule a plan for a single night, it can schedule multiple nights just by running it sequentially. Therefore, with a weather generator it can be used to simulate observing over an arbitrary period of time. In principle, one could give it a very long plan interval and then in one pass it would fill in multiple nights.

It has been generalized to schedule any number of sites together. It evaluates all observations over all sites and schedules the observation with the highest score at the site where it is best to observe it.

While it is trying to maximize the score, which should effectively maximize the metric, it is not currently doing a true mathematical optimization. With appropriate terms in the scoring it selects reasonable observations, though the order could sometimes be improved, and the plans are comparable to those produced by other algorithms. While perhaps good enough, there is no guarantee that the plans will be optimal. There are ways in which the algorithm could be improved, or it could be used in conjunction with other techniques.

A strong advantage to this approach is that it can be tailored to meet most or all of the requirements of the scheduler.

3 Evaluation

Prototypes were written in Python 3 for all proposed techniques except for Spike, which is an independent Common Lisp program.

After running multiple experiments on various data sets and metrics, the basic prototypes that satisfied the most basic of requirements were analyzed using a Pugh table to determine their applicability as candidates to the Gemini automated scheduler.

The first column comprises the requirements, with the second column representing their weight, or relative importance in comparison to the other requirements. Higher numbers represent requirements with higher importance to Gemini.

For each requirement, a score from -2 to 2 was assigned depending on:

- The difficulty of implementing the requirement.
- The ability to satisfactorily implement the requirement.

As a baseline, an actual Queue Coordinator was chosen, and as per Pugh matrix standards, assigned a score of 0 for every requirement. Then, for each possible automated scheduler technique, a score between -2 and 2, measured relative to the Queue Coordinator, was assigned to each requirement.

The final scores were calculated by summing the weight of each requirement times the score assigned to that requirement for the technique under consideration.

Gemini Automated Scheduler Trade Study Pugh Matrix

4 Conclusion

As is visible on the Pugh table, the currently most promising candidate is the GreedyMax algorithm, which was specifically designed with Gemini in mind. In second place are Gurobi ILPs, followed by the genetic algorithms.

GreedyMax in particular meets Gemini's automated Scheduler requirements better than the other techniques in the following areas:

- Speed: GreedyMax runs quickly and is able to calculate plans in real-time, i.e. less than one minute.
- Flexibility: GreedyMax, as developed, allows us to do things like handle AND / OR observation groups and insert calibrations, and adjust the score as it builds up the plan.
- Splitting: GreedyMax can split observations into pieces, which would be difficult with the other techniques considered.

Bibliography

Dueck, Gunter. “New Optimization Heuristics: The Great Deluge Algorithm and Record-to-Record Travel.” *Journal of Computational Physics*, vol. 104, no. 1, 1993, pp. 86-92.

Johnston, Mark D., and Glenn E. Miller. “SPIKE.” *SPIKE: Intelligent Scheduling of Hubble Space Telescope Operations*, 1993, https://www.stsci.edu/~miller/papers-and-meetings/93-Intelligent-Scheduling/spike/spike-chapter3.html. Accessed 01 09 2020.

Kubanek, Petr. “RTS2.” *Remote Telescope System, 2nd Version*, 2008, https://rts2.org. Accessed 01 08 2020.

Lampoudi, Sotiria, et al. “An Integer Linear Programming Solution to the Telescope Networking Problem.” *arXiv.org*, 2015, https://arxiv.org/abs/1503.07170. Accessed 01 08 2020.

Mahoney, William, et al. “A genetic algorithm for ground-based telescope observation scheduling.” *Proceedings of SPIE*, vol. Observatory Operations: Strategies, Processes, and Systems IV, no. 8448, 2012, p. 15. *Proceedings of SPIE*, https://spie.org/Publications/Proceedings/Paper/10.1117/12.926662?SSO=1.