

A Comparison of Techniques for Scheduling Fleets of Earth-Observing Satellites

Al Globus
CSC
NASA Ames

James Crawford
RIACS
NASA Ames

Jason Lohn
NASA Ames

Anna Pryor
NASA Ames

Abstract

Earth-Observing Satellite (EOS) scheduling is a complex real-world domain that is representative of a broad class of over-subscription scheduling problems. Over-subscription problems arise in domains where the requests for a facility exceed the capacity of the facility on a variety of dimensions (e.g., available time, data capacity, power, etc.). Oversubscription problems arise in a wide variety of NASA and terrestrial domains and are an important class of scheduling problems because such facilities often represent large capital investments. We have run experiments comparing multiple variants of the genetic algorithm, stochastic hill climbing, simulated annealing, squeaky wheel optimization and iterated sampling on two realistically-sized models of the EOS scheduling problem. These are implemented as *permutation-based* methods; methods that search in the space of priority orderings for observation requests and evaluate each permutation by using it to drive a greedy scheduler. Simulated annealing performs best and simple mutation operators outperform our squeaky (more intelligent) operator. Furthermore, taking smaller steps towards the end of the search improves performance.

Introduction

A growing fleet of scientific, military, and commercial Earth observing satellites (EOS) circles the globe. Although there are approximately 60 EOS satellites in orbit today, image collection is nearly always scheduled separately for each satellite with manual coordination, if any. Some studies (Globus *et al.* 2002) (Rao, Soma, & Padmashree 1998) have suggested that automatic coordination of multiple satellites can be beneficial, but the best scheduling techniques to use is not clear. The problem is complicated by the fact that EOS scheduling is subject to multiple complex constraints, including power, thermal, data capacity, and the limited time each satellite spends over each target. When we consider the total number of observations that can be performed by a satellite constellations and the number of options (in terms of time windows) there are for each observation, we find that the full search space for EOS scheduling is quite large.

More importantly from a research point of view, EOS

scheduling is one instance of a larger class of *over-subscription scheduling problems*. These problems are characterized by a large number of requests for a scarce resource that must be planned or scheduled subject to a complex set of constraints. In addition to EOS, such problems include scheduling planetary probes and rovers, telescope scheduling, scheduling the deep space network, scheduling wind tunnels or other test facilities, scheduling supercomputers, etc. In general, over-subscription problems arise when requests for a capitially intensive facility need to be scheduled so as to optimize productivity subject to a complex set of operational constraints.

Our work focuses on *permutation-based* approaches to over-subscription problems. The key insight underlying such approaches is that if we could greedily schedule the EOS observation requests in an optimal order then we would produce an optimal schedule.¹ Thus a greedy scheduler allows us to search the space of priority vectors (aka permutations) rather than the space of schedules. This change of representation has two key advantages: First, and most importantly, the greedy scheduler can take any priority vector and produce a feasible (though generally sub-optimal) schedule. This means that we can make local moves, including genetic cross-over operations, without straying into infeasible space (in contrast to methods that search in the space of schedules which must work hard to maintain feasibility, or find ways to evaluate the goodness of infeasible schedules). Second, if there are many possible times at which observations can be scheduled it is often the case that the space of possible permutations is significantly smaller than the space of possible schedules.

Computational scheduling techniques have been applied to the EOS scheduling problem by several authors, including:

1. Sherwood et al. (Sherwood *et al.* 1998) used ASPEN, a general purpose scheduling system, to automate NASA's EO-1 satellite.

¹We should note that proving optimality for a permutation-based method in a domain requires a detailed analysis of the constraints and optimization criterion of the domain as well as the details of the greedy scheduler.

2. Potter and Gasch (Potter & Gasch 1998) described a clever algorithm for scheduling the Landsat 7 satellite featuring greedy search forward in time with fixup to free resources for high priority images.
3. Lamaitre’s group has examined EOS scheduling issues including sharing a single satellite among multiple users and comparing multiple techniques. See, for example, (Lamaitre, Verfaillie, & Bataille 1998), (Bensana, Lemaitre, & Verfaillie 1999) and (Lamaitre *et al.* 2000).
4. Wolfe and Sorensen (Wolfe & Sorensen 2000) compared three algorithms, including the genetic algorithm, on the window-constrained packing problem, which is related to EOS scheduling. They found that the genetic algorithm produced the best schedules, albeit at a significant CPU cost.

This study compares thirteen EOS scheduling techniques on a realistically-sized model problem. In particular, we compare simulated annealing, hill climbing, the genetic algorithm, squeaky wheel optimization, and iterated sampling (ISAMP). In the next section we describe the scheduling problem and our model. A description of the scheduling techniques follows. The nature and results of our computational experiments are then presented along with analysis.

EOS Scheduling Problem

In this section we first describe the EOS scheduling problem as perceived by satellite operators and developers. Then we describe the model of the problem used in this experiment.

EOS scheduling attempts to take as many high-priority observations as possible within a fixed period of time on a fixed set of satellite-borne sensors. For example, the Landsat 7 satellite scheduler is considered to have done a good job if 250 observations are made each day. There are generally far more than 250 observation requests. EOS scheduling is complicated by a number of important constraints. Potin (Potin 1998) lists some of these constraints as:

1. Revisit limitations. A target must be within sight of the satellite; and EOS satellites travel in fixed orbits, usually about 800 km up and 100 minutes per orbit. These orbits pass over any particular place on Earth at limited times so there are only a few observation windows (and sometimes none) for a given target.
2. Time required to take each image. Most Earth observing satellites take a one dimensional image and use the spacecraft’s orbital motion to sweep out the area to be imaged. For example, a Landsat image requires 24 seconds of orbital motion.
3. Limited on-board data storage. Images are typically stored on a solid state recorder (SSR) until they can be sent to the ground.
4. Ground station availability. The data in the SSR is sent to the ground (called SSR dumps) when the

satellite passes over a ground station. Ground station windows are limited as with any other target.

5. Transition time between look angles (slewing). Some instruments are mounted on motors that can point side-to-side (cross-track).
6. Power and thermal control.
7. Coordination of multiple satellites.
8. Cloud cover. Some sensors cannot see through clouds. Not only do clouds cover much of the Earth at any given time, but some locations are nearly always cloudy.
9. Stereo pair acquisition or multiple observations of the same target by different sensors or the same sensor at different times.

Our model problems implements all these constraints except the last two. Both model problem consists of three satellites in Sun-synchronous orbits (orbits in which the equator is crossed at the same local time each orbit) for one week. The satellites are spaced ten minutes apart. Each satellite carries one sensor mounted on a cross-track slewable motor that can point up to 24 degrees to either side of nadir (nadir is straight down) and turns one degree in two seconds. In problem one, each satellite has an SSR capable of storing 50 arbitrary units. In problem two, the SSR stores 75 units.

We model power and thermal constraints using so called duty cycle constraints, the approach taken by NASA’s Landsat 7 satellite. A duty cycle constraint requires that the sensor not be turned on for longer than a maximum time within any interval of a certain length. This insures conformance with power, thermal, and other physical constraints on the spacecraft. Our model problem uses the Landsat 7 duty cycles. Specifically, a sensor may not be used for more than:

1. 34 minutes in any 100 minute period,
2. 52 minutes in any 200 minute period, or
3. 131 minutes in any 600 minute period.

There is one ground station in Alaska. Whenever a satellite comes within sight of the ground station it is assumed to completely empty its SSR, which is then available for additional observation storage. There are approximately 75 SSR dumps per spacecraft during the week. Since some orbits are over oceans and all targets are on land, some SSR dump opportunities are wasted on an empty SSR.

6300 observation targets were randomly generated on land. Of these, 6114 were observable by at least one satellite during the one week scheduling period. The targets are assumed to be at the center of a rectangle that requires 24 seconds of satellite motion to image. Each observation requires one, three, or five arbitrary storage units (evenly distributed) on the SSR. Each observation was assigned a priority from one to six evenly spaced in 0.1 increments. Each observation has 2-24 windows, times when a satellite is within view of the

observation’s target. Orbits and windows were determined by the free version of the Analytical Graphics Inc.’s Satellite Tool Kit, also known as the STK (see www.stk.com).

The fitness (quality) of each schedule is determined by a weighted sum (smaller numbers indicate better fitness):

$$F = w_p \sum_{O_u} P_o + w_s S + w_a A \quad (1)$$

where F is the fitness, O_u is the set of unscheduled observation, P_o is the priority of an observation, S is the total time spent slewing, A is the sum of the off-nadir pointing angle for all scheduled observations, w stands for weight, $w_p = 1$, $w_s = 0.01$, and $w_a = 0.00137$ for problem one and $w_a = 0.02$ for problem two. Note that the weights favor the priority of unscheduled observations over pointing and slewing time objectives, and that the off-nadir pointing objective has very little influence on problem one. w_s is set so that adding an observation always increases fitness, but just barely for a $P_o = 1$ observation. w_a in problem two is set similarly.

There only two differences between the model problems: problem two has more SSR space and the off-nadir pointing objective is much more important. Additional SSR space implies that the duty cycle constraint will become more important.

Scheduling Techniques

This study compares thirteen search techniques applied to the EOS scheduling problem. The simplest techniques were simulated annealing, hill climbing, two variants of the genetic algorithm, and ISAMP (essentially random search) taking random steps. By using a more intelligent mutation operator, these algorithms (except ISAMP) become variants of squeaky wheel optimization (Joslin & Clements 1999).

We represent a schedule as a permutation or arbitrary, non-temporal ordering of the observations. The observations are scheduled one at a time in the order indicated by the permutation. In psuedo-code:

1. int[] permutation = permutation of the integers 1-numberOfObservations
2. for(int i = 0; i != numberOfObservations; i++)
 - (a) schedule observation permutation[i] if it does not violate any constraints

This allows us to search in permutation space (Syswerda & Palmucci 1991) rather than schedule space, as is somewhat more common. DISCUSS A simple, deterministic, one-observation scheduler assigns resources to observations in the order indicated by the permutation. This produces a set of timelines with all of the scheduled observations, the time they were taken, and the resources (SSR, sensor, pointing angle) used. The one-observation scheduler assigns times and resources to observations using earliest-first scheduling heuristics

while maintaining consistency with sensor availability, onboard memory (SSR) and slewing constraints. If an observation cannot be scheduled without violating the current constraints (those created by scheduling observations from earlier in the permutation), the observation is left unscheduled.

Simple earliest-first scheduling starting at $time = 0$ had some problems. We discovered that the algorithm works better if, for each observation, ‘earliest-first’ starts at some random initial time rather than at $time = 0$. This time is, in general, different for each observation. If the observation cannot be scheduled between the initial time the end of time, the algorithm starts at $time = 0$ and continues searching for a constraint-free window until the observation is scheduled or the initial time is reached. The time each observation is scheduled (or, if unscheduled, what time ‘earliest-first’ search started) is stored along with the permutation, is preserved by mutation and crossover, and is used as the starting point for the one-observation scheduler operating on modified versions of the current permutation. The extra scheduling flexibility may explain why this approach works better than earliest-first starting at $time = 0$.

Constraints are enforced by representing sensors, slew-motors and SSRs as timelines. Scheduling an observation causes timelines to take on appropriate values (i.e., in use for a sensor, slew motor setting, amount of SSR memory available) at different times. These timelines are checked for constraint violations as the one-observation scheduler attempts to schedule additional observations.

The simplest algorithm tested was ISAMP, which is essentially a random search. With ISAMP, each schedule is generated from a random permutation with random start times for the one-observation scheduler.

The next class of algorithms tested were the ‘evolutionary’ search techniques, which we define here as those that start with random permutations and generate new permutations with mutation and/or crossover. Unlike ISAMP, these algorithms learn in the sense that they use past experience and gradually improve the schedules generated. The algorithms tested were:

1. Stochastic hill climbing (Hc), which starts with a single randomly generated permutation. This permutation (the parent) is repeatedly mutated to produce one new permutation (a child) which, if the child represents a more fit schedule than the parent, it replaces the parent.
2. Simulated annealing (Sa), which is similar to hill climbing except less fit children can replace the parent with a probability that depends on an artificial temperature. The temperature starts at 100 (arbitrary units) and is multiplied by 0.92 every 1000 children (100,000 children are generated per run).
3. A steady-state tournament selection genetic algorithm (Gs) with population size 100. The individual to replace is chosen by a tournament from the whole

population where the least fit is replaced. Tournament size is always two.

4. A generational elitist genetic algorithm (Gg) with population size is 110 where the 10 best individuals are copied into the next generation. Parents are chosen by tournament (size = 2).

Each search technique was tested with three mutation operators:

1. Random swap (Rs). Two permutation locations are chosen at random and the observations are swapped, with 1-15 swaps (chosen at random) per mutation. Earlier experiments (Globus *et al.* 2003) determined that allowing more than one swap improved scheduling (see Table 3).
2. Temperature-dependent swap (Td). Here the number of swaps (1-15) is still chosen at random but with a bias. Early in evolution a larger number of swaps tend to be used, and later in evolution fewer swaps are performed. This is analogous to the 'temperature' dependent behavior of simulated annealing. The choice of the number of swaps is determined by a weighted roulette wheel where the weights vary linearly as evolution proceeds starting at n and ending at $16 - n$ where n is the number of swaps. Earlier experiments tried fewer swaps early in evolution and more swaps later. This didn't work as well.
3. Squeaky shift (Ss). This mutation operator implements squeaky wheel optimization. The mutator shifts 1-15 (chosen randomly) 'deserving' observations earlier in the permutation. Early in the permutation an observation is more likely to be scheduled since fewer other observations will have been scheduled to create additional constraints. Each observation to shift forward is chosen by a tournament of size 50, 100, 200, or 300 (chosen at random each time). The observation is always chosen from the last half of the permutation. The position-to-shift-in-front-of is chosen by a tournament of the same size (each time) and is guaranteed to be at a location at least half way to the front of the permutation (starting at the 'deserving' observation). The observation most deserving to move earlier in the permutation is determined by the following characteristics (in order):
 - (a) unscheduled rather than scheduled
 - (b) higher priority
 - (c) later in the permutation

The position-to-shift-in-front-of tournament looks for the opposite characteristics.

In the case of the genetic algorithms half of all children are created by mutation and the other half by crossover. The crossover operator is position-based crossover (Syswerda & Palmucci 1991). Roughly half of the permutation positions are chosen at random (50% probability per position). The observations in these positions are copied from the father to the same permutation location in the child. The remaining observations

fill in the child's other permutation positions in the order they appear in the mother.

We have tested a number of other mutation operators but the ones examined in this experiment performed the best. See (Globus *et al.* 2003) and Table 3 for some of these data. We also tested heuristic-biased stochastic sampling (HBSS) (Bresina 1996) with contention heuristics (Frank *et al.* 2002), a technique that operates in schedule space rather than permutation space. HBSS was hundreds of times slower, required far more memory, and produced schedules barely better than ISAMP. There are many techniques that search schedule space and these results are not sufficient to make any sweeping conclusions comparing permutation space and schedule space search.

Experiment

To find the best algorithm for the model problems we compared a total of thirteen techniques. These were ISAMP, and every combination of four search techniques – hill climbing, simulated annealing, steady state GA, and generational GA – with three mutation operators – 1-15 random swaps, 1-15 temperature dependent swaps, and 1-15 squeaky shifts. Thirty-two jobs with identical parameters (except the random number seed) were run for each algorithm. Each job generated approximately 100,000 schedules (the GA runs generated slightly more). On one Athlon processor of our Linux cluster these jobs took 2-3 hours each.

Table 1 compares the algorithms for problem 1 and table 2 for problem 2. In both tables (and the figures) the techniques are ordered by the mean fitness for problem 1. Most, although not all, of the differences were statistically significant by both t-test and ks-test, with confidence levels usually far above 99%. For the most part, the ordering is similar regardless of fitness objective (priority, slewing time, or off-nadir pointing), although there are some exceptions. Table 3 shows similar results with slightly different techniques on a smaller but related problem. We have had similar results on other problems as well.

Simulated annealing is the clear winner for all problems. For problem one hill climbing with temperature depended swaps equals simulated annealing with random swaps, but on problem one simulated annealing always wins. Even hill climbing outperforms both forms of the genetic algorithm and this is true regardless of mutation operator. ISAMP, as one might expect for random search, performed the worst.

For simulated annealing and hill climbing the temperature dependent swap outperforms all other mutation operators, although for the genetic algorithms random swaps outperforms temperature depended swaps. Both random swap and temperature dependent swaps clearly outperform squeaky shifts for all search techniques.

The small standard deviations for all techniques suggests that all runs for a given technique get about the same fitness. Thus, even if the fitness landscape is multi-modal all the minima must be about the same.

Figures 1 and 2, which show the breadth of each fitness distribution over 32 runs, confirms this view. For this reason, we suspect that this problem requires mostly exploitation, rather than exploration, which also explains the poor GA results. Evolutionary change is spread out over the GA populations rather than concentrated on a single individual as for simulated annealing and hill climbing.

The squeaky shift mutator's performance relative to random swaps suggests that it is smart in the wrong way. In preliminary experiments we also tried swapping, rather than shifting, observations and forcing observations to be swapped into certain parts of the permutation (see Table 3). The shift operator performed the best, but still not as well as the random swap mutator. If random outperforms intelligent, then clearly the intelligence is being applied in the wrong way. We do not understand the dynamics of permutation-space scheduling in any fundamental way, and we don't even know if the dynamics are fundamentally similar for different problems. Until a better understanding is reached, the random swap operators – with a decrease in the number of swaps as evolution proceeds – appear best.

Figures 3 and 4 show the effect of changing w_a from 0.00137 in problem one to 0.2 in problem 2. The absolute value of the off-nadir pointing is reduced as the weight increases, and the range of values is greatly reduced, suggesting that the pointing objective is important enough to affect the search.

Summary

We compared thirteen different permutation-space search techniques for scheduling EOS fleets on a realistically-sized model problem. Simulated annealing outperformed hill climbing which, in turns, outperformed the genetic algorithm. Simple random swap mutation outperformed more 'intelligent' mutation. Reducing the number of random swaps as evolution proceeds improved performance further. Although we examined only two, closely related problems here, we have seen essentially the same results on other problems in this class.

An important follow-up to our work would be an equally thorough study of non-permutation methods; that is methods that search in the space of all possible schedules. We examined one candidate, HBSS with contention heuristics, which performed very poorly. We conjecture that the simplicity of local search in permutation space (particularly the fact that we do not need to search in infeasible space) will lead permutation-based methods to dominate on many oversubscription problems. However, this conjecture can only be evaluated by a head-to-head comparison of the best permutation-based and schedule-based search approaches.

Acknowledgements

This work was funded by NASA's Computing, Information, & Communications Technology Program, Advanced Information Systems Technology Program (contract AIST-0042), and by the Intelligent Systems Program. Thanks also to Bonnie Klein for reviewing this paper and to Jennifer Dungan, Jeremy Frank, Robert Morris and David Smith for many helpful discussions. Finally, thanks to the developers of the excellent Colt open source libraries for high performance scientific and technical computing in Java (<http://hoschek.home.cern.ch/hoschek/colt>).

References

- Bensana, E.; Lemaitre, M.; and Verfaillie, G. 1999. Earth observation satellite management. *Constraints* 4(3):293–399.
- Bresina, J. 1996. Heuristic-biased stochastic sampling. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*.
- Frank, J.; Jonsson, A.; Morris, R.; and Smith, D. 2002. Planning and scheduling for fleets of earth observing satellites. In *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics, Automation and Space 2002*.
- Globus, A.; Crawford, J.; Lohn, J.; and Morris, R. 2002. Scheduling earth observing fleets using evolutionary algorithms: Problem description and approach. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*.
- Globus, A.; Crawford, J.; Lohn, J.; and Pryor, A. 2003. Scheduling earth observing satellites with evolutionary algorithms. In *Conference on Space Mission Challenges for Information Technology (SMC-IT)*.
- Joslin, D. E., and Clements, D. P. 1999. Squeaky wheel optimization. *Journal of Artificial Intelligence Research* 10:353–373.
- Lamaitre, M.; Verfaillie, G.; Frank, J.; Lachiver, J.; and Bataille, N. 2000. How to manage the new generation of agile earth observation satellites. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space*.
- Lamaitre, M.; Verfaillie, G.; and Bataille, N. 1998. Sharing the use of a satellite: an overview of methods. In *SpaceOps 1998*.
- Potin, P. 1998. End-to-end planning approach for earth observation mission exploitation. In *SpaceOps 1998*.
- Potter, W., and Gasch, J. 1998. A photo album of earth: Scheduling landsat 7 mission daily activities. In *SpaceOps 1998*.
- Rao, J. D.; Soma, P.; and Padmashree, G. S. 1998. Multi-satellite scheduling system for leo satellite operations. In *SpaceOps 1998*.
- Sherwood, R.; Govindjee, A.; Yan, D.; Rabideau, G.; Chien, S.; and Fukunaga, A. 1998. Using aspen to

automate eo-1 activity planning. In *Proceedings of the 1998 IEEE Aerospace Conference*.

Syswerda, G., and Palmucci, J. 1991. The application of genetic algorithms to resource scheduling. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, 502–508.

Wolfe, W. J., and Sorensen, S. E. 2000. Three scheduling algorithms applied to the earth observing systems domain. *Management Science* 46(1):148–168.

algorithm	F (fitness)	F StdDev	$\sum_{O_u} P_o$ (priority)	$S/ O_u $ (slewing)	$A/ O_u $ (pointing)	$ O_u $ (unscheduled)
SaTd	9205	20	8571	17	10.1	2211
HcTd	9310	21	8659	18	10.3	2289
SaSr	9311	19	8662	18	10.2	2250
HcSr	9368	25	8716	18	10.3	2313
SaSs	9489	19	8872	19	10.4	2583
HcSs	9507	24	8865	19	10.4	2512
GgSr	9700	38	9017	20	10.3	2430
GsSr	9700	25	9019	20	10.4	2430
GsTd	9741	31	9049	20	10.5	2428
GgTd	9834	24	9130	20	10.5	2458
GgSs	9964	53	9281	21	10.5	2652
GsSs	10010	46	9330	21	10.4	2673
ISAMP	10463	11	9727	23	10.7	2723

Table 1: Scheduling algorithms tested ordered by mean fitness for 32 jobs on problem 1. All values are means except column 3. Smaller values are best. Column heading labels refer to equation 1. SA stands for simulated annealing, HC for hill climbing, Gs for steady-state GA, Gg for generational GA, Rs for random swaps, Td for temperature dependent swaps, and Ss for squeaky shifts.

algorithm	F (fitness)	F StdDev	$\sum_{O_u} P_o$ (priority)	$S/ O_u $ (slewing)	$A/ O_u $ (pointing)	$ O_u $ (unscheduled)
SaTd	5571	23	3954	16	9.5	1118
SaSr	5648	22	4009	17	9.6	1125
SaSs	5786	29	4163	18	9.9	1332
HcTd	5870	28	4237	18	9.7	1246
HcSr	5913	36	4273	18	9.6	1258
HcSs	6032	38	4419	18	9.8	1421
GgSr	6306	45	4640	19	10.0	1371
GsSr	6317	44	4646	19	10.0	1375
GsTd	6340	35	4642	19	10.1	1351
GgTd	6489	39	4782	20	10.2	1399
GgSs	6735	66	5088	21	10.2	1615
GsSs	6839	78	5185	21	10.3	1638
ISAMP	7797	12	6124	23	10.6	1774

Table 2: Same as table 1 except data (and ordering) from problem 2.

search algorithm	transmission operators	mean fitness
Sa	1-9 Rs	2171
Sa	1 Rs	2354
Hc 5 restarts	1-9 Rs	2539
Hc 5 restarts	1 Rs	2564
Hc 0 restarts	1 Rs	2575
Sa	1 squeaky swap	2772
Sa	1 placed squeaky swap	2814
Hc	1 squeaky swap	2868
Gs	crossover and 1 Rs	3007

Table 3: Results from a somewhat different problem with a different, but related, set of search techniques. Note that the overall results are similar. Here the problem has two satellites and 4000+ observations with SSR size, slewing range and times, and other aspects different from the model that generated Table 1. Details can be found in (Globus *et al.* 2003).

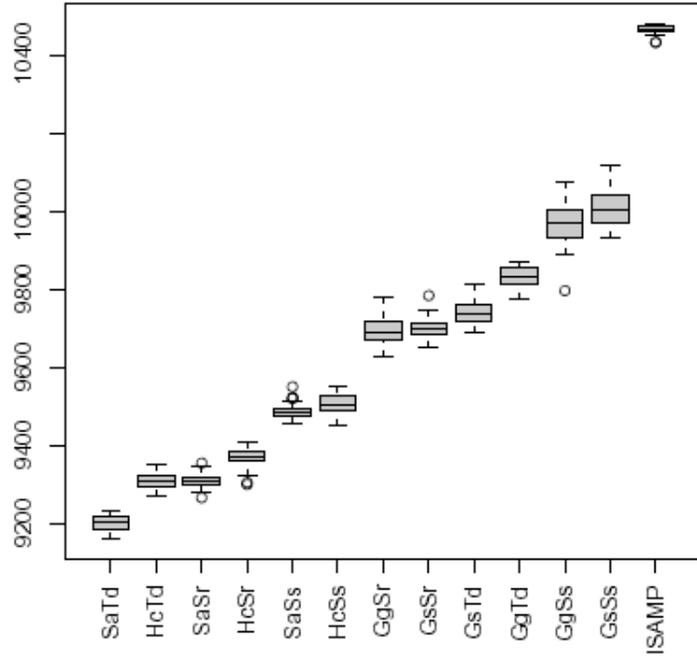


Figure 1: Comparing fitness (vertical axis) for 32 runs for experiment 1. The boxes indicate the second and third quartiles. The line inside the box is the median and the whiskers are the extent of the data. Outliers are represented by small circles. Smaller numbers indicate better fitness.

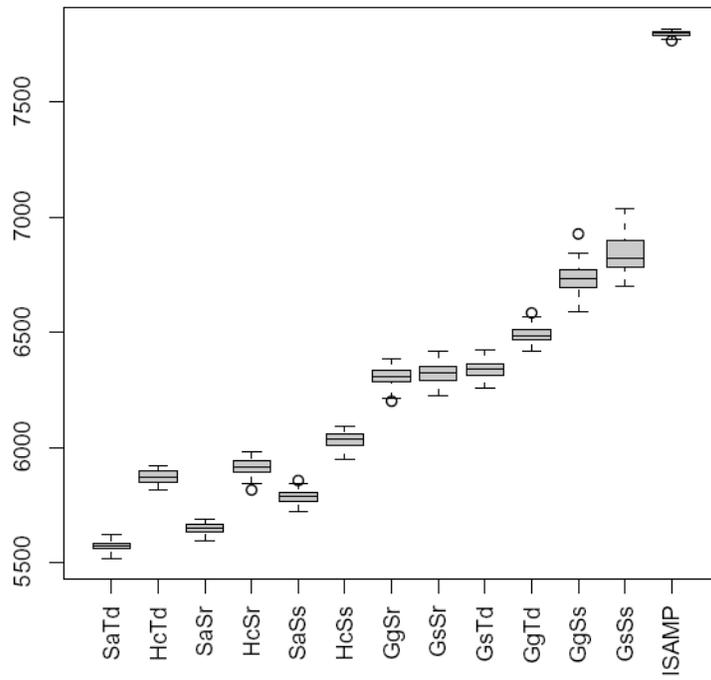


Figure 2: Same as figure 1 with data from experiment 2.

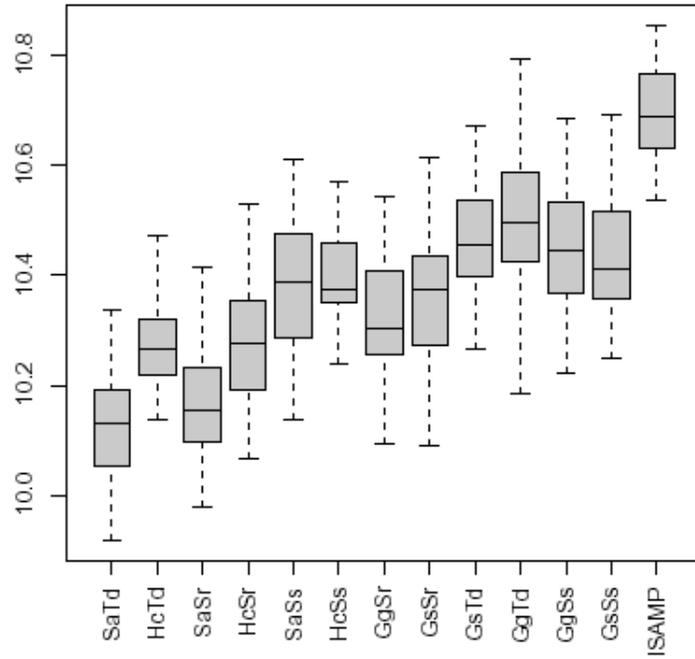


Figure 3: Mean off-nadir pointing angle needed for each scheduled observation (mean of A from Equation 1).

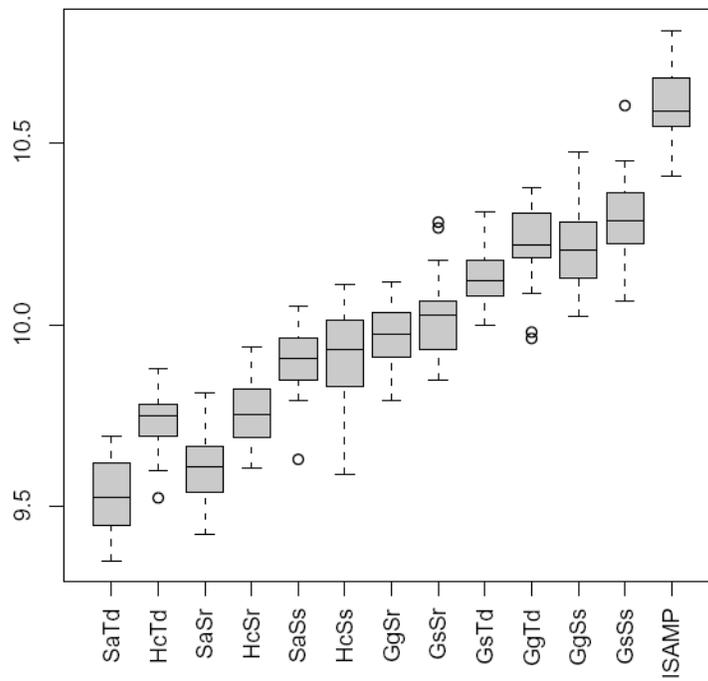


Figure 4: Same as figure 3 with data from experiment 2.