

Evaluation of greedy, genetic and composite scheduling optimization algorithms applied to remote sensing satellites.

Trevor Ackerman, Nicole Nemer, John Clark
Department of Computer Sciences
University of Colorado at Denver
Denver, Co ZIPCODE

Abstract

Two popular strategies for scheduling optimization are greedy scheduling and genetic algorithms. One of the most notable characteristics of tasking a satellite for sensing is the time window that a request needs to be satisfied within. The contention/conflict relationships between the requests drives which scheduling optimization to use. Different optimization algorithms can be used on subsets of requests that do not have contention with the rest of the requests. A composite scheduler presented in this paper finds subsets of requests that have no contention with other requests and uses different scheduling optimization algorithms to the subsets. Whether or not such an approach can be taken is highly dependent on the satellite model and detecting when subsets of requests are truly independent. The particular model used for this paper considered a satellite that had a time cost for collecting an image and a time cost for slewing its camera to point at a request's target.

1 Constraint Based Search Concepts

The constraint based search assigns a set of parameters with certain values for a given domain that satisfies a set of known constraints. The classical example of this is map coloring. Different regions on a map are assigned different colors from a limited choice of colors so that no two neighboring regions are the same color. Solving such a problem means that as soon as the first set of assignments that satisfies the constraints are found the problem is solved and is often referred to as a Constraint Satisfaction Problem or CSP for short. A modification on the CSP is that the problem is not solved as soon as any set of assignments satisfy the constraints. This may be because either it may not be possible to entirely satisfy the constraints, or more than one set of assignments exist that satisfy the constraints but they are not equally desirable. This is known as a Constraint Optimization Problem or COP for short. The classical example of this is the traveling salesman problem. Both CSP and COP are NP hard as each may have to explore as many possible sets of assignments to the parameters to reach their ultimate goal. This could happen for CSP when there is only one possible set of assignments to the parameters given that satisfy the given constraints. It could happen for COP when it is desired to know which set of assignments provide *the* optimal solution.

2 Search Optimization Approaches

Given infinite time and resources most anything can be accomplished. In the spirit of that statement, one could exhaust all possible combinations of values to all parameters to find the desired solution. All search optimization algorithms

presented are concerned with reducing the search space. The two main categories for reducing the search space are constructive algorithms and stochastic algorithms. Constructive algorithms start with a partial solution and use continual evaluation to work towards a complete solution. They often use backtracking to recover from partial solutions that fail to meet given constraints and they often have an upper bound on the amount of backtracking allowed. Many of the classical approaches to the search optimization are constructive approaches including: branch and bound, greedy, and local search algorithms. Stochastic algorithms generate random populations of solutions and then evaluate the fitness of each solution within the population to then generate another population to evaluate. An upper bound is placed on the number of populations to generate. When the upper bound is reached, the most fit solution found at that time is chosen as the solution. Examples of stochastic algorithms are simulated annealing, and genetic algorithms.

3 Evaluation of Search Optimization Approaches

3.1 Constructive Algorithms

Greedy

Greedy algorithms try to maximize the use of some resource. Whatever choice is available that leaves the most useable amount of the resource remaining is chosen next. Then it repeats this tactic until no remaining choices can use the amount of the resource remaining.

Structures known as matroids can be used for finding optimal solutions with greedy algorithms. The matroids represent possible solutions from a set of candidates. Matroids are closely related to minimum spanning trees. If one then adds a weight function to the edges of and has a way to express the total weighted value of each set within the matroid, then a complete solution may be created from an initially empty or partial solution. First the individual edges possible from the graph are weighted and sorted. The algorithm runs in $O(n \lg n + n f(n))$ time total: $n f(n)$ to calculate the weight of each possible edge and $O(n \lg n)$ to do the search. A simple task scheduling problem that schedules fixed discretely timed tasks can be solved with matroids and greedy algorithms [1]. However, the matroid can only be applied to a set of edges whose weights are independent of when they are added to the tree structure.

Dynamic Programming

Dynamic programming has similar goals to greedy algorithms and has the potential for parallelization. However one of the problems is that it does not work so well with nondiscrete values like time. The typical example of where dynamic programming does not work is the fractional knapsack problem. [1]

Evaluation of Greedy and Dynamic programming

Matroids and dynamic programming cannot be used when the previous ordering of variable assignments changes the weight of an unchosen edge. Imagine if the knapsack problem also meant that choosing an item took some variable amount of time depending on the last choice and that the optimal solution was expressed as not just within a weight constraint but also a time constraint. Then the matroids and dynamic programming would not work. This strategy also does not scale well as more constraints and variables are added because each choice would then have to be a representation of possible variable assignments. Still it is possible that a simple greedy sorting heuristic may help in some cases to reduce the search space. This idea will be explored later in the algorithms that were implemented for this study.

3.2 Branch and Bound Algorithms

Branch and bound algorithms seek to optimize the search for a solution by pruning some of the possible choices for the next set of parameter assignments. Without the bound part of branch and bound a partial solution is built up towards a complete solution until either the solution is complete or the partial solution has something added to it that violates the given constraints. Then it merely backtracks continually until it finds a complete solution that meets the constraints or determines no solution exists. The backtracking algorithms could be depth first or breadth first and it is possible to put some amount of randomization into the backtracking. The bound part of branch and bound is the pruning of choices available during backtracking.

Heuristics with branch and bound

Heuristics combined with branch and bound tend to either apply more constraints on the search space or order the choosing of variables or the values in a way to further prune the search space. Examples of heuristics include:

- **Variable Ordering:**
 - Most constrained resource [2] is choosing to assign a value to the variable that is most likely to cause a conflict next. It does a good job on reducing the search space versus plain branch and bound.
 - Edge finding involves a group of constraints that tie a set of tasks to a common resource[3]. An example is a set of tasks that want to use a resource and have different start time windows.

- **Value Ordering:**
 - An example of value ordering is to choose a value that will allow most flexibility in choosing the next variable assignment (similar to a greedy algorithm)[2].

- **Constraint Propagation:**

Forward checking [2] is that after one of the variables is assigned a value from its domain, all the other unassigned variables have the values removed from their domains that would cause a conflict with the newly assigned variable. By itself it does not do as well as most constrained resource, but when combined with most constrained resource it is one of the best heuristics.

- **Retraction/Backtracking:**

Complete backtracking is not always a practical endeavor. Many algorithms using a branch and bound strategy use heuristics for backtracking. One example is that when a conflict is found, to start the search over again from the beginning. Other heuristics add random sampling to their backtracking to increase the breadth of their partial search.

3.3 Local Search

Local search algorithms start with a complete solution but then do a “backwards” constructive search by fixing the choices that have the most conflicts. The classic example is to use a random selection of positions for the N Queens problem and then to minimize the conflicts. [2] Local search algorithms are good at finding a solution but do not do well in terms of completeness or finding optimal solutions.

3.4 Stochastic Algorithms

Stochastic algorithms iteratively generate one or more random complete candidate schedules, measures the fitness, and accept or reject the new candidates based on their predecessors' fitness. How the algorithms generate new populations and decide the candidates to accept and reject is what separates them. All algorithms are bounded by a maximum amount of populations to generate. Examples of stochastic algorithms are as follows:

Hill Climbing

Hill climbing is the simplest. It starts with a randomly generated solution, then iteratively generates new solutions. New solutions that are more fit than previous solutions are kept while others are rejected. Due to the fact that it considers only two schedules at a time and keeping only the best found so far, it tends to have difficulty finding global optima. It often gets trapped in local optima[4].

Simulated Annealing

Simulated annealing is very similar to hill climbing but with one important exception. A solution with less fitness than the current best fitness has some probability of getting propagated forward. This allows simulated annealing to

expand its partial search space and escape local minima[9].

Genetic Algorithms

Genetic algorithms try to apply the principle of propagating best fitness forward based on past results. They generate populations of candidate solutions pick the best two candidates found and then from those two candidates replace a subset of the population with new candidates. Many different variations exist for generating new candidates. For scheduling, the candidate is represented as a permutation of a task ordering. One way to generate new candidates is to take the two best candidates (parents) of the current generation, have a portion of one parent's schedule representation propagated to two new candidates (children) and the children's remaining schedule portions are generated randomly. Usually the children replace the parents in the population, however some variations known as elitism keep one or more parents so the best solution found so far is not lost.

4 Relevance of Schedule Optimization to Satellite Tasking

Satellite tasking is more than just finding a solution to a CSP. The real desire is to find the global optimal schedule (or at least a schedule that is nearly globally optimal) for tasking the satellite. While it is desirable to prune the search space to a reasonable size, depth first styles of searches can become trapped in local optima. It is necessary to allow the search to sample a good amount over the search space. Also scheduling often involves continuous time. Only methods found in the papers referenced that are able to deal with continuous time should be explored. Satellite tasking involves more than a simple optimization of fulfilling requests. Each request requires a plan of action. How plans of actions for individual requests interact with one another greatly affect the optimization of the overall schedule. This adds an element of planning to satellite tasking[3]. The algorithm for searching for the optimal order of request fulfillment should be kept separate from evaluating the feasibility/fitness or a complete or partial schedule. It may be wise to use some form of branch and bound to optimize the schedule of actions needed fulfill a certain order of requests (yet not change that order). Satellite tasking seems to be sensitive to subsets within a complete schedule working together to provide an optimal schedule. Predicting where these subsets occur is unsolved at this time and most heuristics that try to partition the search space in a uniform manner fail to sample the search space effectively.

5 Proposal of Search Optimization Strategies to evaluate for Satellite Tasking

A greedy algorithm should be attempted since it has been shown to work effectively when there is not a high contention between tasks for windows of opportunity.

A genetic algorithm should be tried because it allows for trying different methods of producing new generations of candidates and can be parallelized easier than hill climbing or simulated annealing.

6 Model

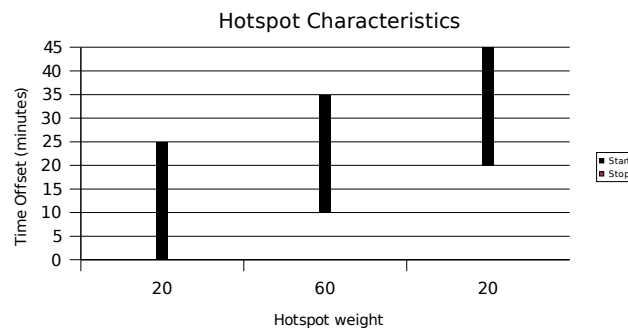
The satellite model will have a fixed latitude and longitude position. It will contain a slewable sensing instrument, characteristics for slewing the instrument as well as the satellite itself. These components are attributes of the satellite. The satellite will be able to sense a target at any latitude and longitude.

A request will be a target, a time window of opportunity, and a value of how desirable it is to fulfill the request. A candidate schedule will be a permutation of requests. A request will be expanded into a course of actions. These actions will be primitive satellite tasks. This will resemble the bridging presented in [3].

Feasibility/Fitness measurement will be kept separate so that it may be used with all search algorithms. The fitness of a proposed schedule will be the summation of the values of all feasible requests.

7 Dataset Generation and Characteristics

Dataset generation was done by first considering an overall time window that represented the time window between uplink/downlink opportunities. Requests were placed in sub-windows within the overall window. These sub-windows were termed hotspots and were given different weights to indicate probabilistically how often a randomly generated request's window should be within that hotspot. Figure 1 illustrates the overall window and the hotspots. Once the dataset was generated it was persisted to long term storage to be reused by all scheduling algorithms. The dataset used to test all schedulers had 175 requests with window time range of 9 to 12 seconds, value range of 100 to 200, slew range of 0 to 10 degrees.



8 Planning Evaluation

Request Feasibility Evaluation

Using ideas from [4] and [3], the satellite model kept time as part of its state. At any given time the satellite model expresses a target it is pointing at and also its current task; idle, slewing, or imaging. All three tasks are mutually exclusive. To fulfill a new imaging request the model requires the sensor to be pointing at the request's target. Planning software was implemented to be able to create a schedule of idle and slewing tasks to meet that constraint. If the planner was unable to create a schedule of idle and slewing tasks to meet that constraint and also could not fulfill the request's time window constraint, the scheduler was notified that the request was not feasible.

Basic Scheduling

A primitive scheduler was created that would take a set of requests and try and build a schedule to fulfill those requests in the order given. Any time that the planner reported a request as not feasible, the scheduler simply discarded that request from the original set given. This scheduler did not evaluate the fitness of the schedule and hence did not try to optimize the schedule given to it. The overall schedule from the requests were the total set of idle, slewing, and imaging tasks for all feasible requests.

9 Scheduling Evaluation

As noted before, the evaluation of any one schedule was the summation of the values of fulfilled requests. All schedulers considered must create their schedule within 30 minutes. Enough storage was assumed to exist to potentially store data from fulfilling all requests. All schedulers used the same set of requests to create their schedules. The schedulers were executed on an Apple iBook G4 with the specifications in the following table.

CPU Type	PowerPC G4 (1.1)
Machine Model	iBook G4
Number Of CPUs	1
CPU Speed	1.2 GHz
L2 Cache (per CPU)	512 KB
Memory	256 MB
Bus Speed	133 MHz
System Version	Mac OS X 10.3.9 (7W98)
Kernel Version	Darwin 7.9.0

Table 1 Test System Characteristics

Greedy Scheduler

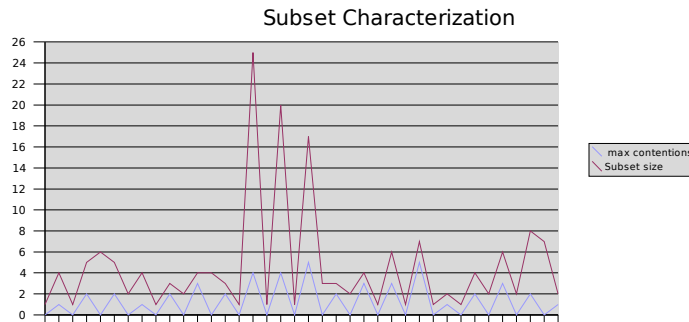
The greedy scheduler achieved its schedule by first reordering the requests by a particular request parameter and then applying the basic scheduler to the reordered requests. Because the time to sort requests and then plan a schedule for them was within one second, the greedy scheduler tried sorting to each parameter and then returned the sorting that produced the most fit schedule. The ordering parameters considered were request window start time, request window finish time, request value, request target longitude. The image request window finish ordering created best schedule and the window start ordering created the second best schedule.

Genetic Algorithm Scheduler

The genetic algorithm scheduler had the following details. Each generation propagated the most fit member forward, also known as elitism. The father chromosome selected was always the most fit chromosome from the previous generation. Selecting the mother chromosome for each new member of a generation was chosen in a statistically weighted fashion. First each member had its fitness measured and then the population was sorted from most to least fit. Next, a random number was chosen from 0 to the summation of all fitnesses. Finally the fitness summation was recomputed and as soon as the summation exceeded the randomly chosen number that member was chosen as the mother. The crossover method was single point crossover, the first half of the father's chromosome was copied then the mother's chromosome was scanned and any value not already in the child chromosome was added. Generation size was 40 members and 20,000 generations were created to produce the final schedule. The mutation rate used was 1 percent. The extremely low mutation rate was chosen because there are few requests that may be swapped without significantly hurting the schedule's optimization. This was similar to results found in [9]. The last generation's most fit member was used as the final schedule. Initial results indicated that the genetic algorithm was going to need a very large search space and hence many generations because it achieved only about 50-60% of the value with 30,000 generations of size 40 members each. The first draft of the algorithm's initialize population routine created entirely randomized permutations of the requests given. Much better results were achieved by incorporating the knowledge from the greedy scheduler into the genetic algorithm when creating the initial population. The first population was created by sorting the requests by window finish time, adding that sorted permutation to the population and then generating the remaining members by repeatedly shuffling the permutation. This guaranteed the genetic algorithm scheduler could do no worse than the greedy scheduler and was used in the final results mentioned below. To achieve a significantly statistical result, the genetic algorithm scheduler was executed 50 times.

Consideration of Request Contention

In the classical problem of map coloring. Any region that has no neighbors cannot be in contention with the rest of the regions and could be assigned any color desired. Because the model allowed for all requests to be fulfilled, it was always probable that some subsets and individual requests would not be in contention with other requests.



The figure shows each independent subset's size and the most number of contentions within the subset. The subsets are ordered by increasing start time. These independent requests and subsets could then be optimized individually to produce the best overall schedule optimization. This provoked the exploration of a composite scheduler.

Composite Scheduler

A composite scheduler was created to take the best features of the greedy scheduler, genetic algorithm scheduler, and additionally an exhaustive scheduler to produce the best overall schedule. The genetic algorithm scheduler by nature explores many request orderings that are not feasible. The greedy scheduler explores a limited number of orderings that are quite feasible to begin with (in particular ordering by window start and finish times). Independent subsets of requests cannot be reordered and maintain feasibility, so these subsets should not be reordered. However the requests within the subsets may be reordered and maintain feasibility and produce an optimal schedule. This is the classic characterization of a greedy problem where optimizing the subsets optimizes the whole set of requests. Also because the subsets vary in size, it is possible to use different schedulers for different subset sizes. Any subset of just one request was left as is, subsets from 2 to 10 were optimized with an exhaustive scheduler, and any subsets of greater size were optimized with the genetic algorithm scheduler. The parameters for the genetic algorithm scheduler were dynamic, the number of generations and generation size were proportional to the subset size. Number of generations was 200 times the size of the subset and generation size was 4 times the subset. The mutation rate was 3 percent which was higher than the original genetic algorithm scheduler because each subset had a better chance of swapping two requests without negative impacts to the schedule. The composite scheduler could produce schedules with fitness at least as good as the genetic algorithm

scheduler at a significantly reduced computing time. The composite scheduler was run 50 times to achieve statistically significant results.

Scheduler Performance Comparison

	Greedy Window Start	Greedy Window Finish	Genetic Algorithm	Composite Window Start	Composite Window Finish
Fitness (deviation or x/y trials)			16164 (0.5%)	16376 (44/50)	16376 (46/50)
Elapsed Time (deviation)	15714 39ms	15816 11ms	581s (2.1%)	16187 (6/50) 244s (1.1%)	16187 (4/50) 244s(1.8%)
Number of executions	1	1	50	50	50

10 Conclusions

The results indicate that the larger a search space is explored on a set of dependent requests, the more optimal a solution can be found. Planning cannot be excluded from scheduling. Planning exposes the additional tasks involved with sensing and their impact on the schedule fitness. The very nature of satellite scheduling is very sensitive and a very small change to the permutation of requests can have very negative impacts to optimizing the schedule. Greedy schedulers may produce a good initial schedule and be very fast, however they often leave room for improvement. Genetic algorithm schedulers produce better results than greedy schedulers but potentially waste many compute cycles on infeasible orderings of requests and also may introduce too much randomness in their search. Because of this potential waste, it is very worthwhile to computationally analyze the contentions between requests. Understanding the contentions between the requests can yield opportunities for applying different scheduling strategies to subsets of requests. This not only saves overall computing time but also allows for strategies such as exhaustive scheduling that would not be practical for the entire set of requests. All the schedulers presented work well continuous time which is often a potential stumbling block for many of the other algorithms surveyed.

11 Further research

The satellite model used was very simple. It had neither limited storage nor did the satellite follow an orbital track. Once storage is limited, the feasibility of collecting images changes drastically. Comparing the fitness of greedy scheduler versus the genetic algorithm scheduler may likely produce more dramatic differences. The greedy scheduler could then be modified to consider a sliding window of subsets or requests that could all potentially be stored. Determining

significant thresholds of storage size versus a dataset size of 175 requests that severely impact schedule fitness would also be a key area of study. Adding an orbital track to the model would be done too. The current fitness evaluation is very simple. Other parameters such as cloud cover and off-nadir angle influence the value of the image and should be included.

Additional schedulers would provide interesting results. Analyzing the contention between requests and understanding can provide good heuristics to use with a branch and bound scheduler. Other papers surveyed used simulated annealing with better success than genetic algorithms and it would be interesting to test the proposed enhancements of the model against both of these stochastic strategies.

Bibliography

- [1] Cormen, Leiserson, and Rivest, Chapters 16 and 17, *Introduction to Algorithms* 1993
- [2] Russel and Norvig, *Artificial Intelligence a Modern Approach 2nd Edition*, Russel and Norvig. Chapter 5
- [3] Smith, Frank, and Jonsson. Bridging the Gap Between Planning and Scheduling, *Knowledge Engineering Review* 2000
- [4] Frank, Jonsson, Morris, and Smith. Planning and Scheduling for Fleets of Earth Observing Satellites, *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space* 2001
- [5] Dungan, Frank, Jonsson, Morris, and Smith. Advances in Planning and Scheduling of Remote Sensing Instruments for Fleets of Earth Observing Satellites, 2002?
- [6] Harrison and Price, Task Scheduling for Satellite Based Imagery, 2000
- [7] Pemberton and Galiber, A Constraint-Based Approach to Satellite Scheduling, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science: Constraint Programming and Large Scale Discrete Optimization* 2001
- [8] Barbulescu, Howe, Watson, and Whitley, Satellite Range Scheduling: A Comparison of Genetic, Heuristic, and Local Search, 2001
- [9] Globus, Crawford, Lohn, and Pryor, Scheduling Earth Observing Satellites with Evolutionary Algorithms, 2003
- [10] Globus, Crawford, Lohn, and Morris, Scheduling Earth Observing Fleets Using Evolutionary Algorithms: Problem Description and Approach, 2003
- [11] Pemberton and Greenwald, On the Need for Dynamic Scheduling of Imaging Satellites, *FIEOS 2002 Conference Proceedings*, 2002
- [12] Morris, Dungan, Frank, Khatib, and Smith, An Integrated Approach to Earth Science Observation Scheduling, 2003