A STUDY OF AIR FORCE SATELLITE ACCESS SCHEDULING

Adele E. Howe, L. Darrell Whitley, Jean-Paul Watson, Laura Barbulescu
Computer Science Department, Colorado State University, Fort Collins, CO 80523 {howe,whitley,watsonj,laura}@cs.colostate.edu

ABSTRACT

Selecting a scheduling algorithm for a given application should be a simple decision. Many algorithms have been proposed and tested, but the testing has mostly been conducted on synthetic benchmark problems for the experimental control they afford. We have found that these results do not necessarily generalize to real applications. To provide the realism lacking in standard testing without sacrificing the experimental control, we have developed a simulator and problem generator for an Air Force application: satellite ground station scheduling. In this paper, we justify our methodology for testing scheduling algorithms, describe the simulator and present some results of algorithm performance on sample problems.

KEYWORDS: artificial intelligence, scheduling

METHODOLOGY FOR EVALUATING SCHEDULING ALGORITHMS

Selecting a scheduling algorithm for a new application should be easy. Scheduling is a well studied area of artificial intelligence and operations research. Countless algorithms have been developed and evaluated. Unfortunately, as advice for developers, the evaluation has two flaws. First, researchers have a tendency toward declaring an all around winner, implying that some scheduling algorithm is the best for all problems. However, theoretical and empirical evidence endorse the view that algorithms tend to excel on specific types of problems, thus complicating the problem of knowing which algorithm to use when.

The second problem is the focus of the evaluation. Commonly, scheduling algorithms have been tested on synthetic benchmark problems. However, in previous research [3], we found that superior performance on synthetic problems does not necessarily translate to real or even realistic applications. Thus, algorithms designed or found to excel on the benchmarks may be inappropriate for most applications.

Clearly, evaluation is needed both to inform developers of which algorithms to use and to promote our understanding of how to create the next generation of algorithms. The obvious option is to evaluate on real problems. Yet, several barriers and concerns preclude this solution. Real problems are difficult to obtain and use. Companies and organizations are often wary of making their data publically available out of concern for what competitors may learn about their internal processes. Additionally, they may not be in a form that matches canonical problems and so require specialization of algorithms and code. From an empirical view, real problems do not often support experimental controls. So it may be difficult to generalize the results of evaluation on such problems.

To gain the benefits of synthetic benchmarks (i.e., large collection of problems and experimental control over the size and structure), we advocate evaluating algorithms with realistic problems derived from problem generators. Such generators differ from previous synthetic generators in that they mimic structure found in real problems and the tunable parameters allow the structure to be systematically varied.

In previous work, we developed problem generators for the well-known flowshop and jobshop scheduling problems [3]. The benchmarks previously used for most studies, Taillard's problems and generator from the OR Library [1], uniformly randomly selected job times for each machine from fixed width distributions. The primary parameters were the number of machines and the number of jobs. In our problem generators, we incorporated two types of structure in these applications: job and machine correlations. A Gaussian distribution of times was constructed for each job or machine, depending on which set of times were supposed to be correlated. The amount of overlap between different distributions is controlled by a parameter; less overlap leads to more problem structure that might be exploited by the scheduling algorithm. We studied the effect of problem structure on algorithm performance and found that some simple algorithms (heuristically guided local search and a stochastic tree search) performed remarkably well in problems with correlation – comparably to the best known algorithm for flow shop problems.

To continue our study and extend our methodology, we have built a problem generator for a new application: satellite remote tracking station scheduling for the Air Force. We recently finished development of the problem generator and an interface for viewing and modifying potential schedules. In this paper, we describe the problem generator and present some preliminary results of performance of some simple heuristic search for solving this problem.

SATELLITE REMOTE TRACKING STATION SCHEDULING (SRTS)

The Air Force has access to its satellites through a small number of geographically dispersed remote tracking stations (nine sites with 16 antennas). Users request access from one of two central sites. The process of scheduling is done manually and is time consuming, requiring about five hours to develop a preliminary schedule. Thus, the human schedulers who solve this problem work about one week in advance of the day they are scheduling; they manage seven sliding windows representing the next seven days of scheduled requests. Requests usually are received this far in advance, but new ones may come in as emergencies arise.

Requests consist of a duration, desired time window(s), ground station(s) and a priority. The requests may be flexible; in fact, often, the duration and priority are inflated by users who might settle for an alternative ground station, shorter duration or a different time window. Typically, about 500 requests are received for each day.

Naturally, more requests are typically received than can be completely accommodated, that is, fit for the entire duration at the correct time on the desired station.

After trying to fit all 500 or so requests into the schedule, often about 120 conflicts remain. At present, the human schedulers must arbitrate the conflicting requests to develop a schedule that maximizes overall request satisfaction.

To solve the problem, the scheduler must find innovative solutions to contention within the physical, practical and priority constraints of the requests. The key issues in automating this scheduling problem are: developing an evaluation measure and incorporating the human scheduler in the loop. The evaluation measure must assign a cost to bumping, moving and reducing requests amortized by the priority of the request. The human schedulers must be included because no feasible schedule is likely to be found most weeks and no program will be able to replace the negotiation process needed to resolve over-constrained resources.

SRTS PROBLEM GENERATOR

The implementation of the SRTS problem generator is based on code from our previous problem generators; the flowshop generator is described in [3]. As per the application description, a problem consists of some number of requests, which, in the simplest case, include a resource, a minimum duration, a maximum duration, a time window, and a priority. The interval midpoint indicates the time at which the request should be half-way completed. To add flexibility to the requests, the generator may also optionally include backup slots for each request.

The problem generator has two types of user-supplied parameters: problem level and request specific. The problem level parameters determine roughly the size of the scheduling instance:

- number of requests,
- number of resources (num-resources), where a resource is a remote tracking station antenna,
- total time (TOTAL) to be scheduled in minutes,
- number of priority levels to be distinguished, and
- minimum and maximum number of backup slots for each request.

A backup slot comprises a resource, time window and minimum and maximum durations.

The request specific parameters define distributions from which to select request values:

- lower and upper bounds on the minimum request duration,
- the maximum duration multiplier (DM), which sets the upper bound on the duration distribution,
- interval width multiplier (IWM), which increases the duration distribution to create time windows in which to schedule the requests,
- number of distinct distributions for time windows.

The first parameter defines the start and end points for the distribution for the minimum durations (min-dur). Likewise, the second defines the start and end for the distribution of the maximum durations (max-dur) as `[min-dur, DM*min-dur]`. The third defines the distribution for the time windows as `[1.0, IWM]`, so a time

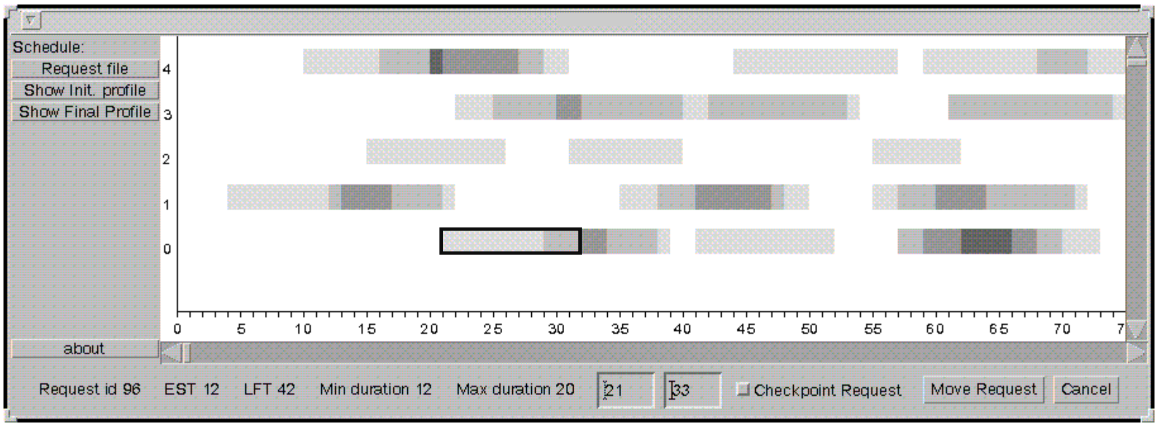

Figure 1: The interface to the SRTS system allows the user to view the initial schedule, examine particular jobs and modify the schedule.

window may be as little as the requested duration but as much as IWM times that duration. Because the time windows may be fit at any position within the total time, we may wish to cluster requests; the fourth parameter indicates how many clusters to create. The remaining values for each request, resource and priority, are sampled from distributions of `[1.0, num-resources]` and `[1.0, max-priority]`, respectively.

For each request, the generator fills in the five basic values by selecting each from the distribution determined by the parameters. Min-dur, resource and priority are sampled uniformly. Max-dur is sampled uniformly from the distribution derived from min-dur; this dependency was intended to model the different types of requests, e.g., short status contacts to check on how the satellite is doing, medium length mission payload commanding contacts, or long maintenance contacts.

The time window is a function of the interval width and the interval midpoint. The interval width (IW) is selected through uniform sampling of its distribution. The interval midpoint indicates the point at which the window is half over. One of two methods may be used to select the interval midpoint. In the first, the distribution, `[.5*IW, TOTAL - .5*IW]`, is unformly sampled. In the second, Gaussian distributions are defined for each resource, each with a different mean and standard deviation. The separate distributions result in clusters of interval midpoints, which simulate regions/times of high demand.

To initialize scheduling, all requests are placed in their most desirable position (centered at the interval midpoint) for their total duration. This results in overlaps. Figure 1 shows an initial schedule, as viewed through the user interface, in which contention is indicated by increasingly dark slots.

TESTING ALGORITHMS ON SRTS PROBLEMS

This application is similar to the well known jobshop problem in that the jobs must be ordered according to their resource requirements and that placing one job implies ordering constraints for other jobs contending for the same resource. It differs from job shop in that each job appears only once on only one resource and that alternative time slots may be considered.

The similarities led us to explore state-of-the-art job shop heuristics and algorithms as solutions to this problem. For our preliminary study, we considered six algorithms: four simple, low cost heuristics as baselines and two based on a state-of-the-art job-shop heuristic. We tested the algorithms on a few problems to determine whether the state-of-the-art algorithm could be easily adapted and whether its additional complexity was valuable in solving this problem.

The Heuristics Tested

The four simple algorithms did little search and used little knowledge.

Maxdur places jobs in the schedule in their desired time window for their maximum duration. If a new request tries to fit into a scheduled slot, discard the request to be added.

Mindur is the same as maxdur, expect that requests are given their minimum time.

Move places requests at their best position with minimum duration initially. Then, it picks the request that has the most overlap with other requests and attempts to resolve the conflict by moving the neighboring requests within their time windows. If the conflict cannot be resolved, then the selected request is removed from the schedule (bumped).

Move+back works the same as Move except that it will consider backup slots as alternatives to simply sliding within the time windows.

The two other algorithms are variants on the min-slack heuristic proposed for the jobshop problem [2]. Each of the algorithms starts from the initial schedule described in the generator section and tries to iteratively improve it by either modifying the placement of requests or removing them entirely from the schedule.

Min-slack selects a pair of requests with the minimum temporal slack between them to order first. Slack is defined for pairs of requests as the minimal difference between the latest finish time of one request and the start time of another minus the processing time for the two requests. When these values are both negative, the schedule is infeasible. In this case, our version removes one of the requests and continues searching for improvements.

Min-slack+back repeats the algorithm of Min-slack except that when an infeasibility is detected one of the two requests is moved to a backup slot. The process is then repeated with what remains in the schedule as requests.

None of the algorithms backtracked. Instead, they would be re-started from the initial schedule; any variance in performance is due to random choice points.

Preliminary Results

We tested the six algorithms on 10 problems of size 600 and counted the number of conflicts (requests that needed to be removed) that resulted. As figure 2 shows, increasing algorithm complexity does produce fewer conflicts. The most dramatic

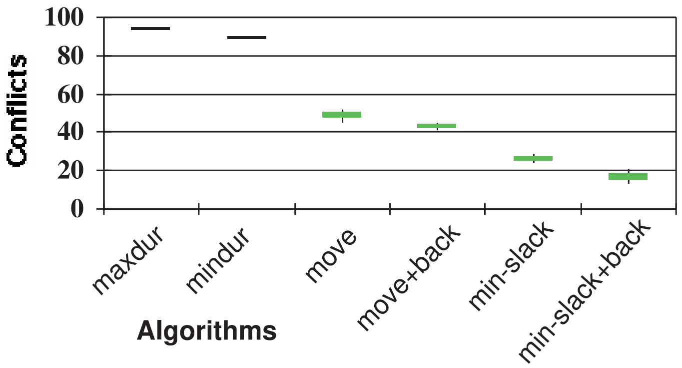

Figure 2: Preliminary results of some simple heuristics on SRTS problems.

reduction is by simply allowing the requests to slide within their time windows. The methods also produce little variance in the results. Clearly, min-slack with backups is the best algorithm for these problems.

FUTURE WORK

We are currently improving both the generator and the solutions. For the generator, we intend to link the determination of the request values so that the resulting requests match likely classes of requests. For example, health contacts should be short in duration, long in time window and low in priority; teleconferencing requests should be long in duration, short in time window and high in priority.

For the solutions, we are further modifying min-slack and other heuristics to better integrate backups and use application knowledge. In addition, we are embedding them in better search procedures to test the contribution of search versus the heuristic.

Acknowledgments

We would like to thank Alex Kilpatrick of AFOSR and Brian Bayless of Falcon Air Force Base for providing us with the information on the application. This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-97-1-0271. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[1] J. E. Beasley. *OR-LIBRARY*. http://www.ms.ic.ac.uk/info.html, 1998.

[2] Steve Smith and C.C. Cheng. Slack-based heuristics for constraint satisfaction problems. In *Proceedings AAAI-93*, pages 139–144, 1993.

[3] J.P. Watson, L. Barbulescu, A.E. Howe, and L. D. Whitley. Algorithm performance and problem structure for flow-shop scheduling. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)*, 1999.