Resource Contention Metrics for Oversubscribed Scheduling Problems

Laurence A. Kramer and **Stephen F. Smith**
The Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh PA 15213
{lkramer,sfs}@cs.cmu.edu

Abstract

We investigate a task insertion heuristic for oversubscribed scheduling problems, *max-availability*, that uses a simple estimate of resource contention to assign tasks to intervals expected to have the best worst case resource availability. Prior research in value and variable ordering heuristics for scheduling problems indicated that sophisticated, but more costly measures of resource contention can outperform simpler ones by more reliably pruning the search space. We demonstrate that for oversubscribed, priority-based problems where a feasible, optimal solution may not even exist, max-availability generates schedules of similar quality to other contention based heuristics with much less computational overhead.

Introduction

In (Kramer & Smith 2003; 2004) we introduced an algorithm *TaskSwap*, as a means of repairing oversubscribed schedules by judicious task retraction and reinsertion. This work demonstrated the efficacy of assigning previously unschedulable tasks by temporarily removing tasks with the highest predicted fitness for reassignment, assigning the problematic task, and then reassigning the displaced tasks. *TaskSwap* is called recursively on any tasks that fail to schedule, succeeding with all tasks being scheduled, or failing when either all contended tasks are tested or a depth bound reached. The setting for this work is the AMC domain(Smith, Becker, & Kramer 2004), a large mission scheduling problem.

Much of this research has focused on the design of retraction heuristics, i.e., what is the best way to predict likelihood of task reassignment when de-assigning a task. However, more recent work (Kramer & Smith 2005b) has demonstrated that impressive gains in solution quality can be achieved by employing resource capacity look-ahead for *task insertion*. The heuristic for task insertion, *max-availability*, has the benefit of being applicable to schedule construction as well as schedule repair and improvement.

Max-availability computes potential resource contention by making the simplest assumption possible, that for each unassigned task 1 unit of capacity will be consumed everywhere in its feasible window. As tasks are assigned, potential capacity is decremented and actual capacity incremented. Potential capacity usage for a resource is the simple aggregation over time of potential capacity usage for individual tasks that can utilize that resource. Where the original *TaskSwap* procedure reassigned retracted tasks at their earliest feasible start, *max-availability* assigns a task on a compatible resource where there is the least predicted contention based on the aggregation of potential and actual capacity.

In this paper we compare the use of this naive metric for potential resource consumption with two more informed measures, *Operation Resource Reliance* (ORR) (Sadeh & Fox 1996)and *SumHeight* (Beck *et al.* 1997). Although these techniques are more computationally intensive, and are not typically applied to oversubscribed problems, our hypothesis was that they might find better solutions in a comparable amount of time due to a better guided search. However, our experimental results demonstrate that for priority-based, oversubscribed problems, even those with structure similar to those most frequently studied by contention-based scheduling research, there is little value added by the use of more sophisticated contention metrics.

The AMC Scheduling Problem

The AMC problem can be described abstractly as follows:

- A set T of tasks (or missions) are submitted for execution. Each task $i \in T$ has an earliest pickup time est_i, a latest delivery time lft_t, a pickup location $orig_i$, a dropoff location $dest_i$, a duration d_i (determined by $orig_i$ and $dest_i$) and a priority pr_i
- A set Res of resources (or air wings) are available for assignment to missions. Each resource $r \in Res$ has capacity $cap_r \geq 1$ (corresponding to the number of aircraft for that wing).
- Each task i has an associated set Res_i of feasible resources (or air wings), any of which can be assigned to carry out i. Any given task i requires 1 unit of capacity (i.e., 1 aircraft) of the resource r that is assigned to perform it.
- Each resource r has a designated location $home_r$. For a given task i, each resource $r \in Res_i$ requires a positioning time $pos_{r,i}$ to travel from $home_r$ to $orig_i$, and a de-positioning time $depos_{r,i}$ to travel from $dest_i$ back to $home_r$.

A schedule is a *feasible* assignment of missions to wings. To be feasible, each task i must be scheduled to execute within its $[est_i, lft_i]$ interval, and for each resource r and time point t, *assigned-cap*$_{r,t} \leq cap_r$. Typically, the problem is over-subscribed and only a subset of tasks in T can be feasibly accommodated. If all tasks cannot be scheduled, preference is given to higher priority tasks. Tasks that cannot be placed in the schedule are designated as *unassignable*.

Task Swapping

In (Kramer & Smith 2003; 2004), a repair-based scheduling approach called *TaskSwap* was introduced as a means of overcoming the limitations of a greedy procedure used to generate mission schedules in the AMC domain. The *TaskSwap* procedure starts with an initial baseline schedule, built under the assumption that higher priority tasks take precedence over lower priority tasks, and a set U of unassignable tasks; and aims at locally rearranging some number of current assignments into the schedule such that 1 or more tasks in U can be feasibly added. This is accomplished by temporarily suspending the assumption that preference be given to higher priority tasks, and using another criterion to resolve the conflicts blocking specific $u \in U$ from being feasibly scheduled.

For a given u, a set of competing tasks is retracted to clear space for scheduling u. When a given task is retracted, the *TaskSwap* procedure attempts to find another assignment for it. For those retracted tasks that cannot be immediately reinserted, the *TaskSwap* procedure is recursively called with this task, resulting in retraction of 1 or more additional tasks. If it is possible to feasibly reschedule all tasks that have been retracted before bottoming out, then this new solution is accepted and attention moves on to the next unassignable task. If, alternatively, it is not possible to reinsert all retracted tasks, then the state of the schedule prior to consideration of u is restored and u remains unassignable. Given a set of conflicted tasks, one question concerns *which* task(s) to retract to create space for u. However, more recent work with the Max-Availability heuristic (Kramer & Smith 2005b) has shown that the decision of *where* to reinsert retracted tasks is actually much more important to the overall performance of the *TaskSwap* procedure.

Max-Availability

Max-availability inserts a task at the start time of an interval with greatest predicted resource availability, estimated using a simple resource "texture" measurement, *max-predicted-availability*.

Estimating Resource Contention

We compute *max-predicted-availability* as follows:

- set the *potential capacity* $PC_{r,i}$ and *allocated capacity* $AC_{r,i}$ to 0 for each resource $r \in Res_u$ to 0 for each time instant $i, 0 \leq i \leq horizon.$
- For each unassigned task, u, set $PC_{r,i} = PC_{r,i} + 1$ for each $r \in Res_u$ and for each $i \in ReqInt_{r,u} = [est_u - pos_{r,u}, lft_u + depos_{r,u}]$.
- For each assigned task, a with assigned resource, r, set $AC_{r,i} = AC_{r,i} + 1$ for each $i \in SchedInt = [st_a, et_a]$.
- For each $r \in Res$, and any subinterval s on r's timeline,
$$Availability_{r,s} = Cap_r - (AC_{r,s} + PC_{r,s})$$

Generating Predicted Availability

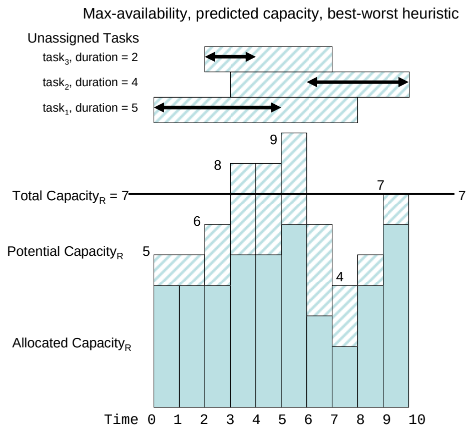

Figure 1: Predicted resource contention

The situation depicted in Figure 1 represents *Availability* for resource R from $time_0$ through $time_{10}$, before inserting 3 tasks. $Task_1$ has a duration of 5, and must be scheduled in the interval [0,8], $task_2$ has a duration of 4, and must be scheduled in [3,10], and $task_3$ has a duration of 2 and must be scheduled in [2,7]. Each task requires 1 unit of capacity.

Resource R has a total capacity of 7 over the interval [0,10] and as indicated in the figure, varying amounts of capacity are currently consumed over this interval. Adding the actual capacity for each interval to the potential capacity, and subtracting from the total capacity (7), we have in the worst case 2 units of available capacity over [0,2], 1 unit available in [2,3], and -1 unit available in [3,5]. The fact that -2 units are available in the interval [5,6] does not make that interval infeasible, since this is a worst case computation.

The Max-Availability Heuristic

The above computation for $Availability_{r,s}$ is used by the *max-availability* heuristic to determine to which resource and in which time interval to assign a task during the scheduling search. The heuristic is implemented as follows: For a task i, and $\forall r \in Res_i$ a linear search is made across the capacity time-line, collecting availability values for subsets of resource feasible subintervals, termed *capacity-subinterval profiles*(Kramer & Smith 2005b). These profiles are of length equal to dur_i and must not cover any values for a subinterval s_r for which the actual capacity $AC_{r,s}$ is 0. For example, in Figure 1 the capacity-subinterval profiles for $task_1$ are (2,2,1,-1,-1), (2,1,-1,-1,-2), (1,-1,-1,-2,1), and (-1,-1,-2,1,3), corresponding to start times of 0,1,2, and 3.

The interval matching the capacity-subinterval profile with the greatest minimal, or maximin, value is selected for assignment. When there is no profile with a unique maximin value, a leximin (Moulin 1988) ordering is used.

Experimental Design

For our experiments with max-availability, we used the 100-problem data set of mission scheduling problems for the AMC domain generated in (Kramer & Smith 2005a). The 100 problems are divided into 5 sets of 20 problems, each set being progressively more resource constrained. For each problem an initial schedule is generated according to strict task priority order, with the max-availability heuristic employed to determine where to insert a task. No back-tracking is allowed, and when the initial pass is complete, some number of tasks that were unable to be assigned may remain. The TaskSwap procedure is applied to all unassignable tasks, employing max-flexibility as the retraction heuristic and max-availability as the commitment heuristic. Run-times in seconds and the end number of unassigned tasks are recorded. As a baseline (Table 1, col. 3), we use the best reported results of Experiment 3 in (Kramer & Smith 2005a). We then substituted differing resource texture measurements for the *max-predicted-availability* metric, in an effort to provide better guidance to the *max-availability* heuristic.

Sophisticated Metrics for Resource Contention

The *max-predicted-availability* texture measurement represents a worse case prediction of resource consumption. By allocating 1 unit of potential capacity for a task over its entire feasible window for every compatible resource, it ignores the fact that the task will eventually be assigned in one window equal to its run duration on one resource. There is a significant body of research that has studied the use of more sophisticated contention measures to guide scheduling search. The aggregate demand *contention* profiles employed by Sadeh's *Operation Resource Reliance (ORR)* heuristic(Sadeh & Fox 1996), and later approximated by Beck's *SumHeight*(Beck *et al.* 1997) texture metric, produce well-grounded probabilistic models of resource consumption.

These models take into account the fact that a task of reasonable duration can only possibly consume capacity in the first interval of a given resource in the case that it is assigned at its earliest start time on that resource, whereas it will consume capacity in the second interval when its start time is at its earliest or when it starts at the start of the second interval. For example, given $task_1$ from Figure 1, a probabilistic model of its resource demand can be represented as seen in Figure 2. For a rigorous discussion of probabilistic metrics for resource consumption, see (Sadeh & Fox 1996).

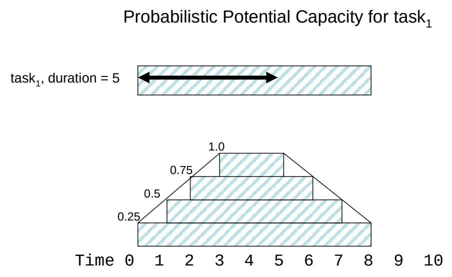
Probabilistic Potential Capacity for task$_1$
task$_1$, duration = 5
1.0
0.75
0.5
0.25
Time 0 1 2 3 4 5 6 7 8 9 10

Figure 2: Probabilistic resource contention

	Set	MaxPredAv	ORR	SumHeight
Tasks remaining	1	0	0	0.05
	2	1.2	1.2	1.15
	3	11.35	11.45	11.35
	4	29.3	28.9	29.4
	5	75.4	75.2	75.45
Time (seconds)	1	0.15	1.45	4.5
	2	2.4	24.95	94.1
	3	20.7	242	2353.65
	4	52.25	707.85	4486.8
	5	206.9	1423	15562.75

Table 1: Quality and Run Times for AMC Problem

Using SumHeight as a Texture Metric

As both Sadeh and Beck point out, computation of resource texture using the aggregate demand profiles of *ORR* is very expensive. Beck introduces an approximation (Beck *et al.* 1997), which relies on computing 4 points corresponding to the end points of the lines along the slopes of the "pyramid" in Figure 2. Interior points can be quickly calculated using a standard point-slope equation. This calculation applies to a single-resource problem domain, but in (Beck & Fox 2000) the authors present a model for scheduling with alternative resources, that effectively scales the *SumHeight* computation by dividing by the number of resource alternatives. We follow this same methodology in our computations, while recognizing that alternative resources are not strictly comparable due to variations in task duration.

We re-ran the baseline 100-problem set, using the *SumHeight* point-slope approximation texture measurement to guide the *max-availability* heuristic. We could not hope to improve on the optimal solutions found for all 20 problems in the first set; however, we expected that for the more difficult problems, *SumHeight* might carve a quicker path through the search space to find better solutions, compensating for the additional time needed to compute predicted capacity. This turned out not to be the case. On average *SumHeight* gave similar results to *max-predicted-availability*. Furthermore, average run-times for the problems were 2 orders of magnitude slower than the baseline. (Table 1, column 5; lower numbers are better.)

Using ORR Rough Demand Profiles

In practice, Sadeh realized that approximations to the *ORR* aggregate demand profiles need be even coarser than those employed by Beck when computation time is at a premium. "Rough demand profiles" are computed by spreading half the task's expected demand evenly across it feasible window and the other half over its most recent best assignment(Sadeh 1994). Using this technique as the texture measurement, we replicated the 100-problem test. Although this turned out to be much faster than the *SumHeight* computations of the prior runs, the experiments still took on the average an order of magnitude longer than the baseline experiments, and demonstrated just a very slight quality improvement in the hardest problems. (Table 1, column 4)

	Set	Baseline	MaxPredAv	ORR	SumHt
Tasks	1	3.95	3.7	3.85	3.45
	2	5.3	4.9	5.15	5.2
	3	5.3	4.95	4.95	4.8
	4	12.1	11.7	12.1	11.1
	5	20.35	20.75	20.05	20.25
Time	1	1.85	2.2	2.3	3
	2	1.95	2.5	2.3	3.4
	3	2.05	2.5	2.25	3.75
	4	4.35	6.4	6.55	8.75
	5	6.05	10.55	8.45	13.2

Table 2: Quality and Run Times for Unit Capacity Problems

Unit Capacity Problems

Much prior work in resource contention based heuristics for scheduling has been conducted on job shop scheduling problems, as opposed to a mission scheduling problem such as the AMC domain. One dominant characteristic of that domain is that many missions can be assigned to a given resource at any time. In contrast, most experimental results for the job shop scheduling problem are cast as unit-capacity problems. It's quite possible that in such a setting, more discriminating contention metrics such as ORR or SumHeight may significantly outperform Max-Predicted-Availability.

To test this conjecture we generated a set of 100 problems using the AMC Problem Set as a seed. We assumed a total capacity of 1 for all wings, and derived problem sets by randomly dropping missions from the original data. In this way the problems continue to be oversubscribed to a fairly high degree, but are more manageable in size. The 100 problems are divided into 5 sets of 20, where the first set is comprised of about 100 randomly selected missions to be allocated, the second set of approximately 200 missions, and so on.

Unit Capacity Experimental Results

To provide a baseline, we first ran all problems using no resource contention metrics for task insertion. A schedule was generated by allocating tasks in priority order at their earliest feasible start times, and the TaskSwap procedure was applied to this schedule. For the second run, Max-Availability – employing the Max-Predicted-Availability texture measurement – was used to guide task insertion during initial schedule construction and the swap phase. The third run substituted ORR Rough Demand profiles as the resource texture measurement, repeating the same procedures. Finally, SumHeight was employed as the texture metric, and the schedule and swap process was repeated a fourth time. As can be seen in table 2 neither SumHeight nor ORR were able to improve appreciably on the simple texture metric in quality. The run times in most cases were slightly higher for SumHeight and comparable for ORR.

Conclusion

Why is the relatively crude resource contention metric, max-predicated-availability, able to compete so favorably with the much more sophisticated measures of SumHeight and ORR? The latter were designed to identify areas of resource contention and particular tasks on which to concentrate the search, and schedule tasks so as to avoid areas of contention to the extent possible. The high degree of computation invested in the search heuristic is expected to pay off by pruning bad decisions as much as it is to guide good ones, reducing backtracking on the path to a problem solution.

The problems we've been studying, though, don't admit to a complete "solution," since they are oversubscribed to start with. A fine-grained estimate of contention is wasted since contention is likely to be high throughout the search space. What appears to make more sense are task insertion heuristics based on easily computed resource contention metrics that are likely to lead to a good solution, as opposed to avoiding bad ones. Furthermore in the problems we've tackled, tasks must be assigned in priority order, so the use of resource contention metrics in variable ordering is of secondary importance in directing the search.

Acknowledgements

The work reported in this paper was sponsored in part by the USAF Air Mobility Command under subcontract 10382000 to Northrop Grumman Corporation, and by the CMU Robotics Institute.

References

Beck, J. C., and Fox, M. 2000. Constraint-directed techniques for scheduling alternative activities. *Artificial Intelligence* 121(1):211–250.

Beck, J. C.; Davenport, A. J.; Sitarski, E. M.; and Fox, M. S. 1997. Texture-based heuristics for scheduling revisited. In *Proc. 14th National Conf. on Artificial Intelligence*.

Kramer, L., and Smith, S. 2003. Maximizing flexibility: A retraction heuristic for oversubscribed scheduling problems. In *Proc. 18th International Joint Conf. on AI*.

Kramer, L. A., and Smith, S. F. 2004. Task swapping for schedule improvement, a broader analysis. In *Proc. 14th Int'l Conf. on Automated Planning and Scheduling*.

Kramer, L. A., and Smith, S. F. 2005a. The amc scheduling problem: A description for reproducibility. Technical Report CMU-RI-TR-05-75, Robotics Inst, Carnegie Mellon.

Kramer, L. A., and Smith, S. F. 2005b. Maximizing availability: A commitment heuristic for oversubscribed scheduling problems. In *Proc. 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*.

Moulin, H. 1988. *Axioms of Cooperative Decision Making*. Cambridge University Press.

Sadeh, N., and Fox, M. 1996. Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence* 86(1):1–41.

Sadeh, N. 1994. Micro-opportunistic scheduling: The micro-boss factory scheduler. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling*. Morgan Kaufmann Publishers.

Smith, S. F.; Becker, M. B.; and Kramer, L. A. 2004. Continuous management of airlift and tanker resources: A constraint-based approach. *Mathematical and Computer Modeling – Special Issue on Defense Transportation: Algorithms, Models and Applications for the 21st Century* 39(6-8):581–598.