Benchmark Problems for Oversubscribed Scheduling

Laura V. Barbulescu and **Laurence A. Kramer** and **Stephen F. Smith**
The Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh PA 15213
{laurabar,lkramer,sfs}@cs.cmu.edu
Telephone: 01 (412) 268-5900, Fax: 01 (412) 268-5569

Abstract

When organizing a scheduling competition one important issue is identifying the test problems for which the participating teams should demonstrate the performance of their approach. Ideally, the results of a competitive evaluation of different approaches should be useful to both scheduling researchers and practitioners. To enable this, we believe that the process of producing problem instances for the competition should focus on two main issues: 1) generating benchmark problems that abstract features from real-world domains, while still providing guidance to solving problem instances from these domains and 2) choosing the real-world domain/domains to be representative for the type of challenges human schedulers are facing. The contribution of our paper is twofold. First, we propose problems with resource oversubscription as good candidates for the competition problems. Second, we present our approach to generating problems by merging the main common features of two similar real-world oversubscribed applications.

Introduction

Most existent scheduling benchmark problem sets are either synthetic, machine-scheduling problems, or are representative of one specific real-world scheduling application. On the other hand, many real-world scheduling applications share various common features. Generalizing the commonalities of a set of chosen domains when producing a benchmark set for a scheduling competition would make it possible to evaluate different approaches in a much broader context than the usual empirical setup for a single application. The results of the competition could then translate into prescriptions of what works well given certain problem characteristics. The main challenges of producing such a benchmark set are: 1) identifying domains that share common features and could be modeled as a more abstract problem description 2) imposing on top of these common features the main features particular to each domain (while ignoring the less relevant domain-specific features) and 3) still being able to find settings of the features that produce instances similar to ones of the original domains.

In oversubscribed scheduling problems the resources available can not accommodate all input tasks and therefore any solution will specify a subset of the tasks to be assigned to resources. We believe that oversubscribed problems would be a good choice for a competition benchmark set for three main reasons. First, oversubscribed problems are typical in a number of important real-world domains: telescope scheduling, satellite scheduling, tracking space objects, scheduling the shuttle payload; such applications have received increased interest in recent scheduling research. Second, as opposed to classical scheduling, oversubscribed scheduling poses additional challenges: identifying the best subset of tasks to be assigned to the existing resources is in itself a difficult optimization problem. Third, many oversubscribed real-world applications share common characteristics and are therefore amenable to being modeled as a more general problem class; we present a working example of how to generate benchmark problems based on two rather similar oversubscribed scheduling applications.

An oversubscribed scheduling application, the management of Earth observation satellites, has in fact been the object of a scheduling competition: the ROADEF 2003 Challenge (French Society of Operations Research and Decision Analisys 2003). While the application is challenging and interesting in itself, the formulation of the objective function is rather specific to this domain: polygons are used to model the surface to be observed by the satellite and a gain associated with the partial acquisition of a polygon is defined such that it favors the termination of the polygons as opposed to dispersal of image acquisitions between polygons. Given this optimization criterion, it is not clear how the results of the competition would transfer to other similar oversubscribed scheduling applications.

In recent research (Kramer, Barbulescu, & Smith 2007c; 2007a; 2007b), we have gained experience with generating problems that draw their features from two oversubscribed real-world scheduling applications which share similar characteristics: the USAF Satellite Control Network (AFSCN) scheduling problem (Barbulescu *et al.* 2006) and the USAF Air Mobility Command (AMC) airlift scheduling problem (Kramer & Smith 2005). Our work was motivated by the fact that the best techniques for solving AMC scheduling problems don't perform well for AFSCN and similarly, the best techniques for AFSCN do not perform as well for AMC scheduling. We hypothesized that certain differences in the problem characteristics for the two domains can account for

such algorithm performance discrepancies: While priority is a hard constraint for AMC scheduling, there is no task priority specified in AFSCN scheduling; also there are differences in terms of resource capacity and temporal flexibility in scheduling a task inside its time window for the two domains. To test our hypotheses, we implemented a problem generator capable of generating both AFSCN-like problems and AMC-like problems, as well as problem instances that share characteristics with both domains but span the range of values for resource capacity or degree of temporal flexibility for tasks.

The balance of this paper describes the commonalities of the two oversubscribed scheduling domains as well as the features that set them apart, then it presents the challenges of generating representative instances for the two domains and introduces our benchmark of 36 sets of problems, followed by a summary of the lessons learned.

Generating Oversubscribed Scheduling Problems

Most existing benchmark problem sets or problem generators for scheduling domains assume that either all tasks can be assigned feasibly given problem constraints, or that the particular constraints themselves, such as deadlines, are relaxable. Often the best solution to these classes of problems involves minimization of makespan. In most real-world domains, though, some of the problem constraints must be violated to achieve a "good" solution. This is true by definition for oversubscribed scheduling problems, where for most instances it is impossible to assign all tasks.

Many problem generators implemented for classical machine scheduling (e.g. flowshop, jobshop, project scheduling) produce random problem instances (Taillard 1993). Obviously this approach does not work well when generating instances for real-world applications; for example, a study of flowshop scheduling has shown that performance on standard benchmark problems did not generalize to performance on problems with realistic structure (Watson *et al.* 1999). Designing a problem generator to produce problem instances similar to the real data for a given application is a difficult task in itself: even after careful consideration of the real data, unexpected interactions of various parameter settings could produce problems that are largely different from the "real" ones.

Generating oversubscribed scheduling problem instances poses additional challenges. Instances with an uncharacteristically low degree of oversubscription, or none at all, as well as instances with a too high degree of oversubscription as compared to the "real" ones are usually not desirable and should probably be filtered as non-representative.

One possible way to preserve the characteristics of the application is implementing a problem generator by "bootstrapping" the real data, where the real instances are deconstructed into reusable pieces and then the pieces are put back together in a way that still makes sense; a small random variation might also be applied to the actual numbers (Engelhardt *et al.* 2001; Kramer & Smith 2005). In this paper, we use a form of this mechanism to produce AFSCN-like problem instances.

Comparing Our Two Benchmark Domains: AFSCN Scheduling versus AMC Scheduling

Our benchmark problems inherit the main characteristics from two oversubscribed scheduling applications. In this section we identify both the common characteristics of the applications as well as their main differences.

In the AFSCN domain, input communication requests for Earth orbiting satellites must be scheduled on a total of 16 antennas spread across 9 ground-based tracking stations. In the AMC domain, aircraft capacity from 15 geographically distributed air wings must be allocated to support an input set of airlift missions.

Despite the application differences, these two domains share a common core problem structure:

- A problem instance consists of n tasks. In AFSCN, the tasks are communication requests; in AMC they are mission requests.
- Each task T_i, $1 \leq i \leq n$, specifies a required processing duration T_i^{Dur}.[1]
- A set Res of resources are available for assignment to tasks. Each resource $r \in Res$ has capacity $cap_r \geq 1$. The resources are air wings for AMC and ground stations for AFSCN. The capacity in AMC corresponds to the number of aircraft for that wing; in AFSCN it represents the number of antennas present at the ground station.
- Each task T_i has an associated set Res_i of feasible resources, any of which can be assigned to carry out T_i. Any given task T_i requires 1 unit of capacity (i.e., one aircraft in AMC or one antenna in AFSCN) of the resource r that is assigned to perform it.
- Each of the feasible alternative resources $r_j \in Res_i$ specified for a task T_i defines a time window within which the duration of the task needs to be allocated. This time window corresponds to satellite visibility in AFSCN and mission requirements for AMC.
- The basic objective is to minimize the number of unassigned tasks.

One principal difference between the domains is the issue of task priority. In the AFSCN domain there is no explicit notion of priority and all tasks are weighted equally. In the AMC domain, alternatively, tasks (missions) are categorized into one of five major priority classes, and task priorities must be respected whenever scheduling tradeoffs are considered - i.e., it is not possible to substitute a lower priority task for a higher priority task even if this choice enables additional lower priority tasks to be inserted into the schedule. This places an additional constraint on the basic objective of minimizing the number of unassigned tasks.

Consideration of the benchmark problem sets that have been published for each of these domains reveals a few additional differences:

[1] Although, for AMC, the actual durations are resource-dependent.

- The size of the AFSCN instances varies between 419 and 483 tasks, while the size of the AMC problem instances is more than double (983 missions).
- Resource capacity for AFSCN varies between 1 and 3; for AMC, it varies between 4 and 37.
- Degree of temporal flexibility (measured as task duration relative to the size of the resource time windows): for AFSCN, approximately one half of the requests in a given problem instance have no temporal flexibility (these are communication requests for low altitude satellites); for the AMC benchmark problems, temporal flexibility is present for all tasks.

Benchmark Problem Sets

To produce a benchmark problem set common to the AFSCN and AMC domains, we start with five AFSCN problem instances, denoted R1 through R5 (Barbulescu *et al.* 2006). These problems are newer and somewhat more difficult to solve than previously studied AFSCN problems. Using these problem instances as a "seed" we build a set of 1,800 problem instances.[2] Our design goals for this set are as follows:

1. The generated problems should span the continuum of characteristics of the AFSCN and AMC domain, some instances being more AFSCN-like, some more AMC-like, and some in between.
2. Very few (if any) of the instances should be solvable by a greedy scheduler, i.e., the generated problems should be oversubscribed, not trivially schedulable.
3. None of the problem instances should be *outrageously* oversubscribed. For instance, a problem instance with 1,000 tasks should not end up with 700 unassignable tasks after the first greedy scheduling pass.
4. The ranges of resource capacity and task slack should be comparable to typical values in the two domains.
5. The problem instances should be grouped into problem sets which exhibit common characteristics, and be large enough in number to produce statistically significant results.

In order to attain these goals, we proceeded in the following way. We first generated 50 AFSCN-like problem instances: ten new instances for each old one in the five-problem seed. The new instances were generated by moving each task's time window later in time by a uniform random choice over an hour time interval. All other factors were held constant. In this way we ended up with 50 new AFSCN problem instances that are very similar to the original ones. This 50-problem set, (1.1 in Table 1), can be considered in some sense a canonical AFSCN problem set. Given different solution techniques, we expect these instances to respond differently only inasmuch as instance R1 was different from R5 in the original domain data. To produce new problem sets from this set, in theory what we would like to do is to vary just one factor, say problem size or resource capacity, to better understand the effect this has on the performance of competing algorithms. In practice, though, it is not always possible to vary only one factor independently and come up with problem instances that conform to our design goals. Increasing problem size without increasing resource capacity can easily result in problems that are too oversubscribed. Likewise, increasing resource capacity without increasing problem size tends to generate instances that aren't oversubscribed. Similar problems arise if we are not careful when varying capacity and slack independently.

[2]This data can be downloaded from http://www.cs.cmu.edu/afs/cs.cmu.edu/project/ozone/www/DITOPS/publications/afscn-amc-benchmarks.zip.

Prob. Set	Avg. Size	Slack df	Capac. cf	Init.Sched.Unassignables $pf = false$	Init.Sched.Unassignables $pf = true$
1.1	443	0	0	34.1	71.2
1.2	886	0	3	127.7	195.6
1.3	1329	0	9	94.8	170.3
2.1	443	0.5	0	25.1	44.3
2.2	886	0.5	3	81.6	121.6
2.3	1329	0.5	9	56.12	106.6
3.1	443	0.5	3	7.4	15.7
3.2	886	0.5	6	27.3	48.9
3.3	1329	0.5	12	47	65.4
4.1	443	0.9	0	11.6	22.6
4.2	886	0.9	3	37.9	65.3
4.3	1329	0.9	9	32.3	45.4
5.1	443	0	5	4.04	13.5
5.2	886	0	8	34.9	69.0
5.3	1329	0	15	47.8	80.5
6.1	443	0.5	5	3.48	6.8
6.2	886	0.5	8	19.7	29.4
6.3	1329	0.5	15	36.8	44.7

Table 1: Description of the problem sets: the size is either similar to the initial AFSCN problems (*.1 sets), doubled (*.2 sets) or tripled (*.3 sets); df is the duration factor, cf is the capacity factor, and pf the priority flag. The average number of unassignables in a greedy initial solution computed for the 50 instances in each problem set is also recorded.

Given these considerations, starting with the AFSCN-like 50 instance problem set, we generated new problem sets by varying one or more of the following factors: problem size (number of tasks), resource capacity (the number of available units of capacity, not the number of resources), slack (temporal flexibility), and task priority, as follows:

- Problem size: For our experiments, we decided to either keep constant, double or triple the size of the initial AFSCN benchmark problems. When the problem size is kept constant, new problems are produced by moving each task's time window later in time by a uniform random choice over an hour time interval. When the size is doubled (or tripled), two (or three) new tasks are generated for each task in the original problem. The new tasks vary from the initial one in terms of time window and possibly duration.
- Slack (temporal flexibility): A duration factor df is used to determine the durations for each new task. Given a task

T_i, $1 \leq i \leq n$ with an initial duration T_i^{Dur}, the new duration is computed as: $T_i^{Dur} * (1 - random(df, 0))$, where $random(df, 0)$ produces a random number between df and zero. For example, if $df = 0.9$, the new task durations can vary anywhere between the initial duration and 10% of the value of the initial duration.

- Resource capacity: Given a resource r with capacity cap_r (in the initial AFSCN benchmark set), a capacity factor cf is used to compute the new capacity of r as: $cap_r + random(cf, 0)$.
- Priority: A priority flag pf determines if task priorities are present in the problem. When pf is true, task priorities are uniformly sampled from $1..5$ (following the five priority classes in AMC).

For our experiments we generated 36 sets of problems, with 50 instances each. 18 of the sets are produced with no task priorities ($pf = false$), and the other 18 are identical but for the addition of task priorities ($pf = true$). The parameters used to generate the sets are shown in Table 1: the second column represents the average size of the problem instance, while the third and fourth columns represent the value of df and cf respectively. Note that problem set 1.1 with $pf = false$ contains the 50 AFSCN-like problem instances, which are similar to the five original ones (same size, slack and resource capacity, varying the time windows for each task). As a measure of the level of oversubscription in the instances for each set, we use a greedy constructor to build an initial schedule for the 50 instances in each set and record the average number of unassignables in columns five and six.

Summary

Fundamental to the development of a scheduling competition is identification and construction of an appropriate set of benchmark problems. Our intent in this paper has been to make two general points with respect to this issue. First, we believe that oversubscribed scheduling applications represent an important, practical class of problems and hence should be considered for inclusion in any general scheduling competition. Second, we believe that an effective approach to generating test problems for a scheduling competition is one that abstracts and consolidates common features from multiple domains. Such an approach leads to problems that capture the essence and structure of a set of related domains without requiring or giving advantage to scheduling algorithms that are engineered to solve specific applications.

Despite this advantage, the generation of test problems that meet this goal remains a significant challenge. From our experience, we can identify the following difficulties:

- Generating oversubscribed problem instances is in general difficult since care must be taken not to produce problems that are easily solvable (and thus not oversubscribed), or so oversubscribed as to be unrealistic. This factor makes it difficult to control input variables independently. For instance, if resource capacity is increased without increasing number of tasks, the result can easily be problem instances that are trivially solvable.
- Understanding what can be abstracted from a real-life model to create a good benchmark set and still have it resemble the real-life domain is not always straightforward. On the other hand, it is helpful to abstract away as many non-essential domain features as possible so that it is not too onerous for other research teams to replicate results.
- There are limits to the size/hardness of the problem instances that should be generated. They should be large and hard enough to demonstrate real-world features and scalability, but not so hard as to be beyond computational resources.

Acknowledgments

This research was supported in part by the USAF Air Mobility Command under Contract # 7500007485 to Northrop Grumman Corporation, by the Department of Defense Advance Research Projects Agency (DARPA) under Contract # FA8750-05-C-0033, and by the CMU Robotics Institute. Any opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the USAF or DARPA.

References

Barbulescu, L.; Howe, A.; Whitley, L.; and Roberts, M. 2006. Understanding algorithm performance on an oversubscribed scheduling application. *JAIR* 27:577–615.

Engelhardt, B.; Chien, S.; Barrett, A.; Willis, J.; and Wilklow, C. 2001. The DATA-CHASER and Citizen Explorer benchmark problem sets. In *European Conference on Planning*.

French Society of Operations Research and Decision Analisys. 2003. http://www.prism.uvsq.fr/∼vdc/ROADEF/CHALLENGES/2003/.

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

Kramer, L. A.; Barbulescu, L. V.; and Smith, S. F. 2007a. Analyzing basic representation choices in oversubscribed scheduling problems. In *Proceedings of the 3rd Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA-07)*.

Kramer, L. A.; Barbulescu, L. V.; and Smith, S. F. 2007b. Searching alternate spaces to solve oversubscribed scheduling problems. Submitted, Journal of Scheduling.

Kramer, L. A.; Barbulescu, L. V.; and Smith, S. F. 2007c. Understanding performance tradeoffs in algorithms for solving oversubscribed scheduling. In *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI07)*.

Taillard, E. 1993. Benchmarks for basic scheduling problems. *European Journal of Operations Research* 64:278–285.

Watson, J.-P.; Barbulescu, L. V.; E., H. A.; and Whitley, D. 1999. Algorithm performance and problem structure for flow-shop scheduling. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*.