Maximizing Flexibility: A Retraction Heuristic 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

In this paper we consider the solution of scheduling problems that are inherently over-subscribed. In such problems, there are always more tasks to execute within a given time frame than available resource capacity will allow, and hence decisions must be made about which tasks should be included in the schedule and which should be excluded. We adopt a controlled, iterative repair search approach, and focus on improving the results of an initial priority-driven solution generation procedure. Central to our approach is a new retraction heuristic, termed *max-flexibility*, which is responsible for identifying which tasks to (temporarily) retract from the schedule for reassignment in an effort to incorporate additional tasks into the schedule. The max-flexibility heuristic chooses those tasks that have maximum flexibility for assignment within their feasible windows. We empirically evaluate the performance of max-flexibility using problem data and the basic scheduling procedure from a fielded airlift mission scheduling application. We show that it produces better improvement results than two contention-based retraction heuristics, including a variant of min-conflicts[Minton *et al.*, 1992], with significantly less search and computational cost.

1 Introduction

Many scheduling domains present problems that are oversubscribed; problems where there are more tasks to be performed over a given time frame than can be feasibly accommodated by available resources. In such problems, it is inevitably necessary to exclude some tasks from the schedule. Hence, a basic objective is to maximize resource utilization (or somewhat equivalently to accommodate as many tasks as possible). However, in many cases, the situation is further complicated. Input tasks are often differentiated by priorty, implying that some tasks are more important than others and if necessary should be included at the expense of others. In the particular domain that motivates this work, for example, task priorities must be rigidly respected. Though it is theoretically possible to trade one higher priority task for a set of lower priority tasks in some circumstances, this is the exception rather than the rule. Thus, the objective is to accommodate as many tasks as possible within this constraint.

Oversubscribed problems present an interesting challenge for constraint-directed search procedures. Oversubscribed problems are not particularly well suited to formulation within a standard backtracking search framework. Hence the application of constructive approaches depends heavily on the ability of search control heuristics to anticipate resource interactions. Repair-based approaches, on the other hand, tend to operate myopically through infeasible intermediate states in hopes of arriving at a better final feasible state. Their effectiveness in the presence of global constraints (such as enforcement of priority) will similarly rely on the ability of search heuristics to effectively focus the repair process.

In this paper, we take a repair-based search perspective of the problem and focus on locally improving an initial solution. We assume that the initial solution generator is priority-driven, and define a controlled, task swapping search procedure for finding and exploiting opportunities to rearrange currently scheduled tasks and incorporate additional tasks that were excluded from the initial solution. To direct the repair process we introduce a novel retraction heuristic, *max-flexibility*, for choosing which tasks to (temporarily) retract to make room for additional, lower priority tasks. Max-flexibility chooses based on a simple measure of the relative temporal flexibility that alternative competing tasks have for feasible re-assignment elsewhere in the schedule.

We evaluate the efficacy of this approach using data and the basic scheduling procedure obtained from a real-world, multi-mission scheduling problem: the day-to-day airlift scheduling problem faced by the USAF Air Mobility Command (AMC). We compare the performance of max-flexibility to a variant of the min-conflicts heuristic [Minton *et al.*, 1992] (adapted to serve as a retraction heuristic) and another more-informed contention-based heuristic.

2 Basic Allocation Procedure

Without loss of generality the AMC scheduling problem can be characterized 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 contracted 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., one 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*. For each unassignable task i, $pr_i \leq pr_j, \forall j \in Scheduled(T) : r_j \in Res_i \wedge [st_j, et_j] \cap [est_i, lft_i] \neq \emptyset$, where r_j is the assigned resource and $[st_j, et_j]$ is the scheduled interval.

Both the scale and continuous, dynamic nature of the AMC scheduling problem effectively preclude the use of systematic solution procedures that can guarantee any sort of maximal accommodation of the tasks in T. The approach adopted within the AMC Allocator application instead focuses on quickly obtaining a good baseline solution, and then providing a number of tools for the end user to selectively relax problem constraints to incorporate as many additional (initially unassignable) tasks as possible[Becker and Smith, 2000; Kramer and Smith, 2002].

The basic allocation procedure used within the AMC Allocator constructs a schedule incrementally; the set of unassigned input missions is first prioritized, and then missions are successively inserted into the current partial scheduling in priority order. The prioritization scheme utilized in the initial step considers the assigned priority $pr(i)$ of each mission i as its dominant ordering criterion. In case of missions of equal priority, secondary criteria give preference to missions with earlier possible start times and smaller overall slack. A given mission i is inserted into the schedule via a search of alternative options. More specifically, for each $r \in R_i$, a set of candidate execution intervals is generated and evaluated, and the highest ranked alternative is selected as the assignment. In the search procedure's basic configuration, only feasible intervals are generated and the evaluation function emphasizes choices that minimize $pos_{r,i}$, $depos_{r,i}$ and execute i as early as possible. If a mission cannot be feasibly assigned, it is marked as unassignable.

This scheduling process allows for assignment of individual missions on the order of milliseconds, with a two to three week window of approximately a thousand missions assignable in a few seconds.

Since it is generally expected that several or more missions will be unable to be assigned during the first pass allocation process, the user can direct the system to automatically explore the space of constraint relaxation options on a given mission, or can do this on a more interactive basis. Available relaxation options include delaying a mission beyond its due date, over-allocating beyond contracted capacity on a wing, bumping a lower priority mission, or some combination of these basic options. Specifically tailored evaluation functions are used when searching in the space of relaxed constraints. For instance, when searching for delay options, the dominant optimization criterion is minimizing tardiness. [1]

3 Improving on the Initial Solution

The quality of any schedule produced by the above greedy procedure will be a function in large part of its prioritization heuristic. This heuristic dictates the assignment possibilities that will be available to a given task i at the time it is inserted into the schedule, and consequently which tasks will ulitmately end up as unassignable. As just indicated, this heuristic gives over-riding preference to higher priority tasks. This bias in fact reflects the basic scheduling policy in the AMC application domain.

At the same time, strict relance on this ordering heuristic can obviously lead to sub-optimal solutions. It is quite possible that some "rearrangement" of the assignments of higher priority tasks could enable the feasible insertion of additional, lower priority tasks. Since, in practice, the end user may spend non-trivial amounts of time analyzing and negotiating constraint relaxation options to enable incorporation of additional unassignable missions, it makes sense to consider techniques for productively broadening the search performed by this basic procedure.

Two broad classes of approaches have been pursued in the literature. One set of approaches (e.g., [Joslin and Clements, 1998; Bresina, 1996; Cicirello and Smith, 2002]) focuses on exploring a "neighborhood" around the trajectory of the base heuristic. In our context, this would correspond to repeatedly perturbing the task order in some manner and reapplying the basic search procedure. Another set of approaches have been termed *iterative repair* (e.g., [Minton *et al.*, 1992; Zweben *et al.*, 1994; Rabideau *et al.*, 1999]), wherein an initial base solution is progressively revised (and hopefully improved) over time. In our context, this would correspond to repeated retraction and (re)assertion of subsets of task assignments. Both sets of approaches have natural anytime properties. One interesting requirement in the current context, however, is that we would like to guarantee that higher priority missions won't be supplanted by lower priority missions. In the case of iterative re-solving approaches, enforcement of this constraint would seem possible only in a rather

[1] A separate search procedure designed to identify and exploit opportunities for reclaiming resource capacity by combining (or "merging") two or more roundtrip missions into one is also provided, but this capability is orthogonal to the techniques discussed in the current paper.

indirect way (through global filtering of the solutions generated). Iterative repair approaches typically involve generation of infeasible solutions and are thus potentially well-suited to controlled exploration of solution improvement opportunities. However, they are generally not designed to enforce global solution constraints.

Accordingly, the approach we propose takes the solution improvement perspective of iterative repair methods as a starting point, but manages solution change in a more systematic, globally constrained manner. Starting with an initial baseline solution and a set U of unassignable tasks, the basic idea is to spend some amount of iterative repair search around the "footprint" of each unassignable task's feasible execution window in the schedule. Within the repair search for a given $u \in U$, criteria other than task priority are used to determine which task(s) to retract next, and higher priority tasks can be displaced by a lower priority task. If the repair search carried out for a given task u can find a feasible rearrangement of currently scheduled tasks that allows u to be incorporated, then this solution is accepted, and we move on to the next unconsidered task $u^{'} \in U$. If, alternatively, the repair search for a given task u is not able to feasibly reassign all tasks displaced by the insertion of u into the schedule, then the state of the schedule prior to consideration of u is restored, and u remains unassignable. Conceptually, the approach can be seen as successively relaxing and reasserting the global constraint that higher priority missions must take precedence over lower priority missions, temporarily creating "infeasible" solutions in hopes of arriving at a better feasible solution.

In the subsections below, we describe this task swapping procedure, and the heuristics that drive it, in more detail.

3.1 Task Swapping

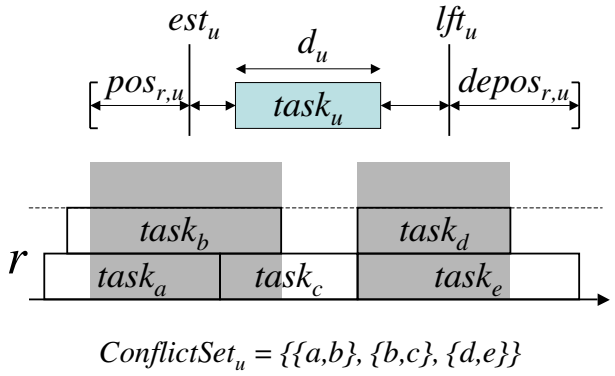

Figure 1: An unassignable task u

Figure 1 depicts a simple example of a task u that is unassignable due to prior scheduling commitments. In this case, u requires capacity on a particular resource r, and the time interval $ReqInt_{r,u} = [est_u - pos_{r,u}, lft_u + depos_{r,u}]$ defines the "footprint" of u's allocation requirement. Within $ReqInt_{r,u}$, an allocation duration *alloc-dur*$_{r,u} = pos_{r,u} + d_u + depos_{r,u}$ is required. Thus, to accommodate u, a subinterval of capacity within $ReqInt_{r,u}$ of at least *alloc-dur*$_{r,u}$ must be freed up.

To free up capacity for u, one or more currently scheduled tasks must be retracted. We define a conflict $Conflict_{r,int}$ on a resource r as a set of tasks of size Cap_r that simultaneously use capacity over interval int. Intuitively, this is an interval where resource r is currently booked to capacity. We define the confict set $ConflictSet_u$ of an unassignable task u to be the set of all distinct conflicts over $ReqInt_{r,u}$ on all $r \in R_u$. In Figure 1, for example, $ConflictSet_u = \{\{a,b\},\{b,c\},\{d,e\}\}$.

MissionSwap($task, Protected$)
1. $Protected \leftarrow Protected \cup \{task\}$
2. $ConflictSet \leftarrow$ ComputeTaskConflicts (task)
3. $Retracted \leftarrow$ RetractTasks ($ConflictSet, Protected$)
4. **if** $Retracted = \emptyset$ **then** Return($\emptyset$) ; failure
5. ScheduleTask($task$)
6. ScheduleInPriorityOrder($Retracted$, least-flexible-first)
7. **loop for** $(i \in Retracted \wedge status_i = unassigned)$ **do**
8. $Protected \leftarrow$ MissionSwap($i, Protected$)
9. **if** $Protected = \emptyset$ **then** Return($\emptyset$) ; failure
10. **end-loop**
11. Return($Protected$) ; success
12. **end**

RetractTasks($Conflicts, Protected$)
1. $Retracted \leftarrow \emptyset$
2. **loop for** $(OpSet \in Conflicts$ **do**
3. **if** $(OpSet - Protected) = \emptyset$ **then** Return($\emptyset$)
4. $t \leftarrow$ ChooseTaskToRetract($OpSet - Protected$)
5. UnscheduleTask(t)
6. $Retracted \leftarrow Retracted \cup \{t\}$
7. **end-loop**
8. Return($Retracted$)
9. **end**

Figure 2: Basic MissionSwap Search Procedure

Given these preliminaries, the basic repair search procedure for inserting an unassignable task, referred to as **MissionSwap**, is outlined in Figure 2. It proceeds by computing $ConflictSet_{task}$ (line 2), and then retracting one conflicting task for each $Conflict_{r,int} \in ConflictSet_{task}$ (line3). This frees up capacity for inserting $task$ (line 5), and once this is done, an attempt is made to feasibly reassign each retracted task (line 6). For those retracted tasks that remain unassignable, **MissionSwap** is recursively applied (lines 7-10). As a given task is inserted by **MissionSwap**, it is marked as protected, which prevents subsequent retraction by any later calls to **MissionSwap**.

In Figure 3, top-level **InsertUnassignableTasks** procedure is shown. Once **MissionSwap** has been applied to all unassignable tasks, one last attempt is made to schedule any remaining tasks. This step attempts to capitalize on any opportunities that have emerged as a side-effect of **MissionSwap's** schedule re-arrangement.

The driver of this repair process is the retraction heuristic **ChooseTaskToRetract**. We define several possibilities in the next subsection.

InsertUnassignableTasks($Unassignables$)
1. $Protected \leftarrow \emptyset$
2. **loop for** $(task \in Unassignables)$ **do**
3. SaveScheduleState
4. $Result \leftarrow$ MissionSwap($task, Protected$)
5. **if** $Result \neq \emptyset$
6. **then** $Protected \leftarrow Result$
7. **else** RestoreScheduleState
8. **end-loop**
9. **loop for** $(i \in Unassignables \wedge status_i = unassigned)$ **do**
10. ScheduleTask(i)
11. **end-loop**
12. **end**

Figure 3: InsertUnassignableTasks procedure

3.2 Retraction Heuristics

In designing a retraction heuristic, our general goal is to retract the task assignment that posseses the greatest potential for reassignment. One simple estimate of this potential is the scheduling flexibility provided by a task's feasible execution interval. More precisely, let *avail-dur*$_i = lft_i - eft_i$ represent the amount of time available for executing task i. Then a simple, resource-indepedent measure of flexibility is

$$\frac{d_i}{\textit{avail-dur}_i}$$

However, recall that d_i is not the total amount of time that the supporting resource must be allocated for. d_i only accounts for the time required to execute the task; it does not account for the time to position and deposition the resource for task execution. The total time that a given resource r must be allocated for is *alloc-dur*$_{r,i} = pos_{r,i} + d_i + depos_{r,i}$. Given that the resources being allocated are generally scheduled near to capacity, a worst-case, resource-dependent measure of flexibilty is

$$\frac{\textit{alloc-dur}_{r,i}}{\textit{avail-dur}_i}$$

Using this notion, we define an overall measure of i's temporal flexibility as

$$Flex_i = \frac{\sum_{r \in R_i} \textit{alloc-dur}_{r,i}}{\textit{avail-dur}_i \times |R_i|}$$

and the following retraction heuristic:

$$MaxFlex = i \in C : Flex_i \leq Flex_j \forall j \neq i$$

where $C \in ConflictSet_u$ for some unassignable task u.

An alternative approach to estimating a task i's potential for reassignment is to measure the level of contention across its required execution interval $ReqInt_{r,i}$. Assume as before that $ConflictSet_i$ is the set of conflicts within $ReqInt_{r,i} \forall r \in R_i$. Then, a very basic measure of contention for a given task i is simply the number of conflicts, i.e. $|ConflictSet_i|$. For purposes of choosing which task to retract in the case of a conflict $C \in ConflictSet_u$, this measure leads to a "min-conflicts" like heuristic[Minton *et al.*, 1992]:

$$MinConflicts = i \in C : |ConflictSet_i| \leq |ConflictSet_j| \forall j \neq i$$

A potentially more informed measure of contention is one that considers the proportion of a task i's required execution interval on resource r that is currently unavailable (i.e., contained in a conflict). For any conflict $C_{r,int} \in ConflictSet_i$, assume that dur_C designates the duration of int. Then we define a task i's overall contention level as

$$Cont_i = \frac{\sum_{C \in ConflictSet_i} dur_C}{\sum_{r \in R_i} ReqInt_{r,i}}$$

Using this measure, we can define a third retraction heuristic:

$$MinContention = i \in C : Cont_i \leq Cont_j, \forall j \neq i$$

4 Computational Analysis

The genesis of the max-flexibility heuristic and the mission-swap algorithm took place while experimenting with the canonical data set used to test and demonstrate the AMC Allocator. This data set, which we'll refer to as the Tutorial Data Set, consists of 982 actual missions (3,251 operations) and 12 actual air wings, and represents a two to three week horizon of AMC airlift and air refueling missions.

Using the basic allocation procedure the system is able to feasibly allocate all but two of the 982 missions in the Tutorial Data Set. For demonstration purposes those two missions are usually added to the schedule by over-allocating a given wing.

It turned out that this particular data set was not as resource over-constrained as we had thought. Application of the mission-swap algorithm with even a random retraction heuristic is able to schedule the two unassignable missions in a few seconds.

4.1 Experimental Design

For our experiments we generated data sets using the Tutorial Data Set as a seed: five data sets of twenty problems each were generated, with the wing capacities randomly reduced from 0 to 10%, 0 to 20%, 0 to 30%, 0 to 40%, and 0 to 50%. For each data set, the basic allocation procedure was employed to quickly schedule as many missions as possible in priority order. The number of unassignable missions was recorded, and then **InsertUnassignableTasks** was executed on the set of unassignable missions. Then run-time, nodes searched (number of times **MissionSwap** was called), and final number of unassignable missions were recorded. For each run this process was repeated with min-conflicts, min-contention, max-flexibility and random choice as the retraction heuristic (**ChooseTaskToRetract**, in Figure2).

Experiments were run on a 1.8Ghz Pentium IV PC with 1Gb of RAM, running Windows 2000. The scheduling engine is implemented in Allegro Common Lisp 6.1.

4.2 Results

The results of our experiments are shown in Figures 4, 5, and 6. As expected, as the random degree of over-allocation was increased from 10% to 50%, the number of initial unassignable missions increased, as did the numbers of unassignables after application of **InsertUnassignableTasks**. Over all problem sets the baseline random-choice heuristic

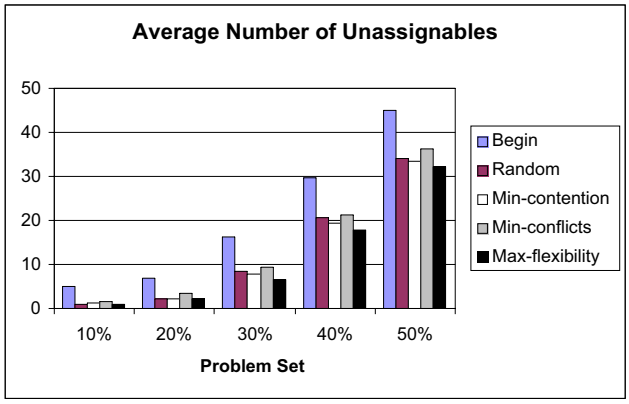

Figure 4: Solution Quality

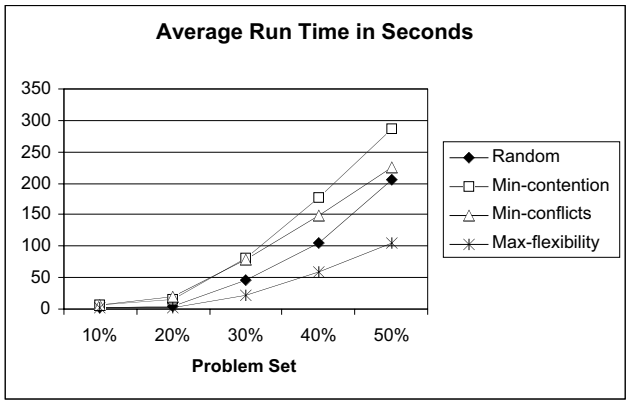

Figure 5: Computational Cost

was able to assign 36% of the unassignable missions. Min-conflicts, min-contention and max-flexibility were able to assign 30%, 38%, and 42%, respectively.

What is most striking, though, are the results shown in figures five and six. While max-flexibility achieved somewhat better results in solution quality, it did so while searching far less and reaching a solution far more quickly than any of the competing heuristics.

4.3 Discussion

Our preliminary results show that in the face of a resource constrained scheduling problem, the decision of what tasks to temporarily retract in order admit more tasks into the schedule may be delegated to an extremely simple and cheap retraction heuristic: max-flexibility. In order for this heuristic to be effective it should retract tasks that have a high likelihood of reassignment elsewhere within their feasible windows.

It is somewhat counter-intuitive, then, that max-flexibility is able to perform so well, based only on task flexibility to reschedule, irrespective of other competing tasks in the schedule. Min-contention performance approaches that of max-flexibility, but only at significant additional cost – approximately triple the run-time on average in our experiments.

Our conjecture is that as a schedule becomes more and more highly constrained, contention will increase to the limit of available capacity, and thus its measure will serve as a less informed heuristic. Under these conditions, though, a metric based on a task's "innate" flexibility to schedule along a timeline might be more useful.

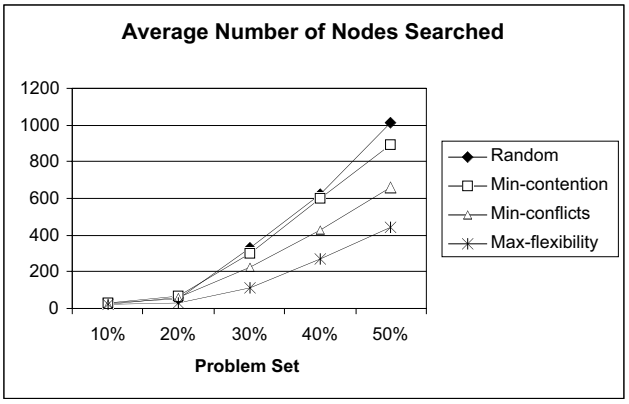

Figure 6: Search Nodes Explored

5 Related Work

The use of measures of temporal flexibility and resource contention as guidance for variable and value ordering has a long history in the field of constraint-directed scheduling [Sadeh, 1991; Smith and Cheng, 1993; Beck, 1999], although for the most part use has been in constructive search contexts. Our work, alternatively, seeks to exploit these measures in an iterative repair search context, to determine which tasks to retract and reassign.

Work in manufacturing scheduling domains has addressed a broad range of priority based scheduling problems[Morton and Pentico, 1993], and some of this work (e.g., [Smith, 1994]) discusses heuristic techniques for schedule repair. However, a broad assumption that underlies most of this work is that due dates are relaxable and the objective is to minimize tardiness. As such, these are not over-subscribed problems in the same sense as the type of problem considered here.

Research in the domain of space mission planning and scheduling, alternatively, has focused on the solution of over-subscribed problems (e.g., [Minton *et al.*, 1992; Johnston and Miller, 1994; Rabideau *et al.*, 1999]). Generally this research is aimed at solving the single mission, single resource problem (e.g., the observing schedule for a space telescope). [Zweben *et al.*, 1994]'s shuttle ground processing domain is one exception, wherein both multiple resources and multi-capacity resources are considered. The approach employed here is repair-based, and simple heuristics are employed to mitigate resource constraint violations. It is not clear how useful this technique would be, however, in addressing problems where maintaining task priority is crucial.

Recent work in the area of scheduling observations on multiple earth satellites comes closest to tackling the same issues which we address: multiple resources, multi-capacity resources, fixed time windows, and mission priority. [Pemberton, 2000] proposes a solution to preserving priority in the

face of over-constrained resources by segmenting the problem into priority classes, solving them individually, and recombining them. [Frank *et al.*, 2001] propose a contention-based heuristic for use in solving a very similar problem within an iterative sampling framework. However, no experimental analysis is given.

Finally, work in constraint satisfaction problem solving (CSP) has explored ideas similar to those introduced in this paper. [Verfaillie and Schiex, 1994] describe a repair procedure similar to **MissionSwap** for non-disruptively resolving conflicts in dynamic CSPs (actually motivated by previous work on a continuous, single-resource scheduling problem). [Prestwich, 2001] utilizes a "largest domain first" retraction heuristic to drive an incomplete backtracking procedure.

6 Conclusions

We have presented a novel retraction heuristic, max-flexibility, and shown its applicability to a multi-mission airlift scheduling problem. This heuristic is much faster and less memory intensive than min-conflicts and min-contention, and provides better solution quality. We suggest that max-flexibility as used in a generic task swapping algorithm like mission-swap may generally be applicable to resource-constrained scheduling problems with fixed time windows. This method is particularly suitable to dynamic scheduling domains, where an existing schedule must be preserved as tasks change and new ones are added.

It is possible that with further study and experimentation a more informed retraction heuristic than max-flexibility might be found. It is likely, though, that such a heuristic would be much more expensive, and that a heuristic that is fairly smart, but very cheap, may be the best choice of all.

Acknowledgements

The work reported in this paper was sponsored in part by the Department of Defense Advanced Research Projects Agency (DARPA) and the US Air Force Research Laboratory under contracts F30602-00-2-0503 and F30602-02-2-0149, by the USAF Air Mobility Command under subcontract 10382000 to Northrop-Grumman Corporation, and by the CMU Robotics Institute.

References

[Beck, 1999] J.C. Beck. *Texture measurements as a Basis for Heuristic Commitment Techniques in Constraint-Directed Scheduling*. PhD thesis, Dept. of Computer Science, University of Toronto, 1999.

[Becker and Smith, 2000] M.A Becker and S.F Smith. Mixed-initiative resource management: The amc barrel allocator. In *Proc. 5th Int. Conf. on AI Planning and Scheduling*, pages 32–41, Breckenridge CO, April 2000.

[Bresina, 1996] J. Bresina. Heuristic-baised stochastic sampling. In *Proceedings 13th national Conference on AI*, pages 271–278, Portland OR, 1996. AAAI Press.

[Cicirello and Smith, 2002] V. Cicirello and S.F Smith. Amplification of search performance through randomization of heuristics. In *Proc. 8th Int. Conf. on Principles and Practice of Constraint Programming*, Ithaca NY, Sept 2002. Springer-Verlag.

[Frank *et al.*, 2001] J. Frank, A. Jónsson, R. Morris, and Smith D.E. Planning and scheduling for fleets of earth observing satellites. In *Proc. 6th Int. Symposium on AI, Robotics and Automation for Space*, 2001.

[Johnston and Miller, 1994] M.D. Johnston and G. Miller. Spike: Intelligent scheduling of hubble space telescope observations. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann Publishers, 1994.

[Joslin and Clements, 1998] D.E. Joslin and D.P. Clements. Squeakywheel optimization. In *Proc. 15th National Conference on AI*, Madison WI, July 1998. AAAI Press.

[Kramer and Smith, 2002] L. Kramer and S.F Smith. Optimizing for change: Mixed-initiative resource management with the amc barrel allocator. In *Proc. 3rd Int. Workshop on Planning and Scheduling for Space*, Houston, Oct 2002.

[Minton *et al.*, 1992] S. Minton, M.D. Johnson, A.B Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1):161–205, 1992.

[Morton and Pentico, 1993] T.E. Morton and D.W. Pentico. *Heuristic scheduling Systems*. John Wiley and Sons, 1993.

[Pemberton, 2000] J.C. Pemberton. Toward scheduling over-constrained remote-sensing satellites. In *Proceedings 2nd Int. NASA Workshop on Planning and Scheduling for Space*, San Francisco, CA, March 2000.

[Prestwich, 2001] S. Prestwich. Local search and backtracking vs non-systematic backtracking. In *Proc. AAAI Fall Symposium on Using Uncertainty within Computation*, North Falmouth, MA, Nov 2001.

[Rabideau *et al.*, 1999] G. Rabideau, R. Knight, S. Chien, A. Fukanaga, and A. Govindjee. Iterative planning for spacecraft operations using the aspen system. In *Proc. 5th Int. Sym. on AI, Robotics and Automation for Space*, 1999.

[Sadeh, 1991] N. Sadeh. *Look-Ahead techniques for Micro-Opportunistic Job Shop Scheduling*. PhD thesis, Dept. of Computer Science, Carnegie Mellon University, 1991.

[Smith and Cheng, 1993] S.F Smith and C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proc.11th National Conference on Artificial Intelligence*, pages 139–144, Wash DC, July 1993. AAAI Press.

[Smith, 1994] S.F. Smith. Opis: A methodology and architecture for reactive scheduling. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.

[Verfaillie and Schiex, 1994] G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proc. 12th National Conf. on AI*, Seattle WA, Aug 1994.

[Zweben *et al.*, 1994] M. Zweben, B. Daun, E. Davis, and M. Deale. Scheduling and rescheduling with iterative repair. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann Publishers, 1994.