Enhancements to the Ground Processing Scheduling System

Juan Jose Blanco and Lina Khatib
Computer Science
Florida Institute of Technology
Melbourne, Fl. 32901
{jblanco,lina}@cs.fit.edu

Abstract

This paper describes work in progress to enhance the performance of the Ground Processing Scheduling System (GPSS). The GPSS is a constraint-based scheduler that deals with three kinds of constraints: temporal, resource, and configuration. It starts with a complete schedule that may have some constraint violations (conflicts) and gradually improves the schedule by reducing the violations. The objective is to optimize the schedule in terms of constraint violations and resource utilization.

The enhancement is accomplished by improving heuristics currently used for constraint satisfaction and adding an extra feature which we call fencing. Fencing allows for obtaining a conflict-free subschedule within a selected period of time.

1 Introduction

The Ground Processing Scheduling System (GPSS) is a constraint-based scheduler. It is used for assigning *times*, and *resources* to all the activities involved in the maintenance, repair and preparation of the fleet of Space Shuttles from the time one lands on the ground until it is next ready to launch. The problem domain is a very demanding one in which activities are constrained by resource limitations, configuration requirements, and temporal specifications.

Due to the over-constrained nature of the domain, which eliminates valid schedules, the problem is approached as a Constraint Satisfaction Optimization Problem (CSOP). In general, the main objectives are to minimize the schedule length while maximizing the resource utilization and satisfying all temporal relations. GPSS uses a hill-climbing method coupled with a simulated annealing implementation to repair conflicts. Temporal consistency is achieved as a first step of every iteration and constraint violations are then resolved gradually, one by one, to improve the schedule. Currently, the annealing algorithm accounts for the possibility of escaping local minima encountered in the search space.

Although the deployment of GPSS has been mainly a success, we have identified certain areas that could be enhanced. Through such enhancements, we provide a way to test techniques applicable towards solving general constraint satisfaction and scheduling problems. In this paper, we propose the following three improvements to the GPSS.

1. Implementing a fencing capability that would allow the user to obtain a conflict-free schedule for a selected period of time.
2. Enhancing the selection of tasks to move when resolving conflicts (constraint violations). This involves a comparative study of computationally cheap local heuristics versus more time consuming look-ahead methods.
3. Modifying the stochastic search algorithm to a greedier one which will only jump out of its current path when a local minima has been reached. This is done by eliminating the random walk aspect of simulated annealing and utilizing less probabilistic approaches to escape local minima.

The remainder of this paper contains a description of the current method for automatic scheduling used in GPSS, the proposed enhancements and their extendibility to more general domains, and finally a summary and future work.

0-8186-7937-9/97 $10.00

2 Automatic scheduling in GPSS

Classical scheduling is the process of assigning times and resources to the activities of a plan. A scheduling problem can be naturally mapped into a constraint satisfaction problem (CSP), where the values (times) that the variables (activities) can take are constrained by temporal relations and by the finite capacity of the available resources. The complexity of classical scheduling is greatly increased when the problem is extended to include configuration requirements and effects, i.e., certain activities requesting an attribute of an object to be in a specified state and modifying such a state when they are completed.

The Ground Processing Scheduling System (GPSS) is a very large constraint-based scheduling assistant developed by Lockheed and NASA to aid in managing the maintenance, repair, and preparation of the Space Shuttle for a new mission. An orbiter processing flow between missions takes, on average, between 1000 and 2000 tasks to be scheduled, for about 10,000 to 16,000 individual shifts of work, and with around 100 scheduling changes made each day [1]. The sheer size of the problem is further complicated by the need of accounting for configuration constraints, as well as by a demanding domain.

The temporal reasoning involved is a two-fold process involving, first, resolving *predecessor-successor* relations between tasks,[1] and second, performing a specialization of classical scheduling called *fixed preemptive scheduling.* Fixed preemptive scheduling implies making activities comply with a *work calendar* which indicates when an activity can be scheduled. Each activity is preempted when it falls outside its work calendar. Thus, an activity is split into the shortest sequence of contiguous subtasks which are all legal with respect to the parent activity's work calendar [10]. The shortest contiguous sequence restriction distinguishes fixed from flexible preemptive scheduling, where idle time (legal with respect to the work calendar) between the split subtasks of an activity is allowed.

All of the above account for the high degree of complexity of the scheduling domain faced by GPSS. Such complexity was the main obstacle the original designers of the system encountered when pursuing a solution based on constructive, or systematic, constraint-solving algorithms. These algorithms could require prohibitive amount of time to solve easy instances of the problem [11]. The intractability and other characteristics of the domain (such as high dynamism and requirement of rescheduling, over-constrainedness, search for optimality) led to the utilization of a repair-based, or stochastic approach. Such an approach provides the following benefits.

1. It offers a solution to the combinatorial explosion that inhibits the performance of systematic CSP algorithms.
2. It attempts to improve solutions as much as possible regardless of whether the problem is over-constrained or not, and terminates with a set of assignments to the variables of the problem that is as close to the solution as could be derived in the time allotted.
3. By searching through the space of complete schedules, it makes "global" optimization criteria possible and cheap to evaluate.
4. In rescheduling cases (very common in highly dynamic domains), it is capable of taking advantage of near-optimal solutions previously derived, which may only need a slight modification to fit the new conditions.

These benefits were major concerns in the environment at hand; and hence, an iterative approach was finally chosen over a constructive one. A more detailed description of CSP techniques falls outside the scope of this paper. For further discussion on systematic versus stochastic CSP algorithms see [2].

2.1 Specification

In this section we describe the specification of the scheduling problem handled by the GPSS [1]. The operation takes place at the Kennedy Space Center Orbiter Processing Facility (OPF).

- Input:
  1. A set of tasks T where each element t includes
     - work duration required to complete t
     - work calendar
     - resource requirements
     - configuration requirements
     - configuration effects
     - release and due dates
  2. A set of temporal constraints C relating activities to one another.

[1] In this paper "tasks" and "activities" are used interchangeably since the distinction is very deep and beyond the scope of the paper

3. A set of resource objects R and their availability profile.
4. A set of attributes A and their initial state.

- Output: A start and end time for each task t such that
  1. no resource is over-utilized at any time
  2. all tasks have their configuration requirements satisfied
  3. all release and due dates are met

2.2 GPSS as CSOP

Due to the over-constrained nature of the problem, which eliminates valid schedules according to the above specification, the problem is approached as a Constraint Satisfaction Optimization Problem (CSOP). The GPSS users search for schedules as good as possible according to certain optimization criteria specified by KSC management. In general, the main objectives are to minimize the schedule length while maximizing the resource utilization, with more and more constraints being relaxed as launch time approaches (e.g., task work calendar can be extended to include overtime and holidays as legal periods).

2.3 GPSS Process

The GPSS provides a graphical interface for the user to specify all input. Then, it performs a first stage of temporal reasoning by satisfying all temporal constraints among tasks. The Waltz algorithm [9], was modified to propagate temporal constraints and achieve temporal constraint satisfaction with $O(N^2)$ complexity, where N is the number of tasks.

As a second step of temporal reasoning, each activity is broken into subtasks according to their work calendar. This setting of tasks serves as the initial assignment for the iterative algorithm to repair. Although the assignment is complete, it is not by any means valid since it contains resource and configuration conflicts. The GPSS provides an interactive mode that allows users to manually resolve constraint violations by moving tasks along the time line. Temporal consistency is maintained after each move by running the Waltz algorithm, and resource and attribute profiles are continuously updated and available to the user through graphical output. The goal here is to enhance the scheduling engine that performs automated conflict resolution and optimization.

GPSS uses a hill-climbing method coupled with a simulated annealing implementation to repair conflicts. The user can select relevant resources to focus on and determine the maximum time allotted for deconfliction. Also, the user can choose to fix tasks in their current time slot. During each iteration of the repair algorithm, the following steps are taken:

1. Select a conflict (resource or configuration) to be resolved taking into account specified priorities.
2. Build a table of tasks involved in the unsatisfied constraint (selected conflict). For each task, calculate ten heuristic values that will aid in determining which task to move for resolving the conflict.
3. The weighted sum of all heuristic values serves to select the task to move forward or backward (following a typical hill-climbing approach). The Waltz algorithm is run to satisfy all temporal constraints. Note that although the move resolves the conflict at hand, running Waltz to achieve temporal consistency again may cause new resource and/or configuration constraint violations to arise.
4. Simulated annealing [3] is then used to obtain the next state. If the move has caused an overall reduction in the number of conflicts, it is accepted and the new state becomes the initial state for the next iteration. But, unlike pure hill-climbing techniques which reject moves to higher-cost states, such moves are made in GPSS with a certain probability to avoid the tendency of stochastic CSP methods of becoming stuck in local minima.

If d is the increase in cost of the new state, a move which will normally be rejected for having a higher cost will now be accepted with a probability $e^{-d/T}$, where T is the temperature parameter which is initially set to a higher value and is then gradually decreased according to a cooling schedule. The probability of moving uphill is therefore directly proportional to the value of T, and inversely to the increase in cost; the worse the new state is, the lower the probability to move to it, but high values of T can make even these moves possible. Simulated annealing basically performs the function of introducing random noise in the search procedure to avoid local minima.

In GPSS, the cooling schedule is very simple, setting the temperature initially to 75 and lowering

it to 25 when a certain cost level is reached. Although simulated annealing typically finishes when the temperature reaches 0 and the search "freezes" in a local minima, GPSS uses the user-fixed maximum time to continue improving the schedule until that limit is reached. It then returns to the user the best answer obtained so far.

3 Enhancement of the GPSS scheduling engine

This section describes the enhancement we propose for the GPSS which affects the efficiency of its automatic deconfliction process.

The deployment of GPSS into its operational environment at KSC has been mainly a success since it was first used in March 1992 for space shuttle Columbia's STS-50 flight. However, as its user community has expanded and has relied on its results for delivering daily schedules for the shuttle ground processing, there has been a clear trend toward utilizing GPSS's interactive mode rather than the automatic scheduling engine. The main reasons for this behavior follows.

1. GPSS global improvement of schedules deviates from the way human schedulers perform their work. Rather than being interested in an overall good schedule extending over the approximately 60 to 80 days a typical flow lasts, the high complexity and dynamism of the environment makes a human expert want a conflict-free schedule for a selected period of time (e.g., schedules are currently updated and delivered to the OPF to account for the next 48 hours of work). GPSS does not allow such preference. Instead, the automatic deconfliction process selects conflicting tasks randomly, from within the whole time-line, and resolves conflicts in each iteration. Hence, the algorithm may be spending a lot of time to resolve constraint violations at the tail end of the schedule which is currently of much less interest to the user. A side effect of the difficulty for human users to appreciate the "goodness" of such complex and global schedules is the loss of confidence in the computer's result.

2. The automatic scheduling algorithm is quite slow which is partially due to the choice of data structures and implementation language of the current version of GPSS. Resolving conflicts in such a complex environment is a time consuming process for any algorithm, but we believe some improvements can be made to the scheduling engine to increase its efficiency. As a first step, the current version is being re-engineered and ported from Lisp to C++.

Following the intention to resolve the above problems, we propose a series of enhancements to the current implementation of GPSS.

3.1 Fencing capability

Adding a feature that allows the user to specify a temporal window of interest on which to focus will greatly increase the applicability of GPSS. Fencing techniques are useful and applicable to any other scheduling system with similar characteristics. The user could select the time period for which he/she desires to obtain a conflict-free schedule and GPSS should then concentrate on removing the conflicts occurring during that period.

The way we intend to introduce this fencing is as follows: once the user has selected the time window over which deconfliction should take place, we will modify the scheduling algorithm so that it will only select conflicts to resolve among those happening within the fencing period. Since our intention is to optimize the partial schedule within the specified time range, we need not only resolve all conflicts in this range, but also maximize resource utilization during that period. We will add a new heuristic to aid in choosing the task to move that will indicate its likeliness to be moved outside the window (easily computable by looking at its temporal constraints).

Once all constraint violations in the time window are resolved, a second stage of the process will be added to maximize resource utilization. This procedure will browse through the under-allocated resources in the time window and will repeatedly try to move back tasks that make use of them without causing other conflicts. Since the problem at hand is over-constrained, maximizing daily resource utilization while satisfying temporal and configuration constraints can actually be seen as an optimal solution to the problem which is easier and achievable in less time than a global one since the algorithm will only be working with a small subset of the problem at a time. Once the window of interest is resolved, GPSS could be left running to improve the remaining schedule as much as possible. This will give the user a long-term idea of at which point constraints will need to be relaxed and/or resource pools augmented.

In the context of CSP solving, fencing can be viewed as the gradual addition of variables to already found solutions to subproblems. In addition,

backtracking to deep levels will not be necessary because the new variables have only constraints with other variables at or near the leaves in the current solution. Therefore, this technique could later be extended to solve complex CSPs whose constraint networks have a quasi treelike structure. In [8] a similar idea is introduced for solving the problem of scheduling a remote sensing satellite. Extending previous solutions with new tasks is successfully performed by making a few local changes to the old tasks that conflict with the new ones.

3.2 Improvement of activity selection for deconfliction

GPSS resolves constraint violations (either resource or configuration) by moving a task or a set of tasks involved in the conflict to a new time period where the conflict does not arise. There are basically two trends in performing the selection of the task to move:

1. Use local heuristics. This is computationally cheap but uninformed in the sense that they are mere heuristics. This type of selection is used by GPSS. GPSS builds for each task involved in a conflict a table with heuristics such as fitness (how close the task's resource requirement is to the amount of over-allocation), temporal dependents (amount of temporally related tasks), and cost (number of conflicts of the new states resulting from a forward and a backward move of the task, without re-satisfying temporal dependencies). This table is then used to select the best task to be moved.

2. Use a greater depth of repair or look-ahead knowledge. For instance, the actual use of heuristics in GPSS does not take into account the fact that moving a task will generally require moving several other temporally dependent tasks when the Waltz algorithm is reapplied. This may cause an increase in constraint violations which has not been previously calculated, and which may be worse than for other tasks that scored lower in the heuristic evaluation.

It is not clear which of the methodologies is better, since performing the more informed look-ahead involves a computational overhead that may degrade the scheduler if the cost is overly expensive. However, this more knowledge-intensive technique assures that each of the moves performed will actually yield the best possible improvement in cost [4].

We intend to run tests in both ends of the scale. First, we will make the calculation of the heuristics table cheaper by eliminating the cost entry. The latter involves deallocation of all the resources for the task, re-allocation of resources at the new time slot, and computation of the violations with the new placement. This entry will be replaced by another, computationally cheaper parameter giving the rate of violated constraints over total constraints involving the candidate task. The task with a higher rate should be considered before others;otherwise it will be harder to resolve conflicts later on.

We will also examine the results of choosing the activity to move based on a full one-step look-ahead (i.e., including performing temporal satisfaction after each move) of a forward and a backward move for all tasks involved in the conflict, selecting among them the one that yields a new state with the lowest cost. Based on these studies, we will determine the optimal technique for activity selection when deconflicting in the Space Shuttle domain, extracting general conclusions about the tradeoff between informedness versus speed in CSP and scheduling problems.

3.3 Enhancement of the stochastic search method

The main advantage of simulated annealing over other stochastic search methods (pure Hill-climbing, Min-conflicts, GSAT) is its ability to escape local minima while searching, instead of having to restart from a new initial assignment. However, precisely the way simulated annealing performs this jump-out has been lately questioned by several authors. For instance, in [6], a performance comparison between a greedy GSAT and the annealing algorithm led to the conclusion: "much of the effort expended by simulated annealing in the initial high temperature part of the schedule is wasted".

High values of T causes the annealing algorithm to wander through the search space without any focus in reducing the cost of the solution. Suppose the current cost is 10 and the cost of the new state is 20. The increment of cost is 10, 100% higher than the current one. In GPSS, the probability of accepting such a move is 0.88, which is very high. Although this might seem to indicate a poor choice of values for the parameter T and/or the need of a more drastic cooling schedule, this totally random walk that simulated annealing performs makes very little sense. In [7], the annealing algorithm was tested for several cooling schedules, yielding no improvement over a range of constant-temperature sched-

ules. The best results were obtained for a low value of T. A close examination of this result reveals that it makes more sense to make the search as greedy as possible, focusing on improving the cost and leaving only a slight probability (low T) of jumping out to escape local minima.

Given that the benefits that simulated annealing brings are doubtful and there is no formal guide to its application, we propose to apply in GPSS a totally greedy algorithm to improve the cost of the current solution when possible, and to rely on a procedure to escape local minima only when one is reached. A variant of the breakout method [5] is one such technique we intend to experiment with. When at a local minima, the breakout method favors a jump to a worse neighboring solution with the smallest number of current no-goods (conflicts). Since the current path has exhausted all possibilities to resolve the current constraint violations, it seems reasonable to try a different path with as few of those hard conflicts as possible. We will compare this method with the simplest, cheapest jump-out technique, used by other stochastic search methods such as tabu search. When at a local minima, where all neighbors of the current assignment are worse than the current one, this technique will jump out to the best of those neighbors.

Both of these noise strategies have the inherent danger of producing too little a perturbation in the search path, making the algorithm fall into the same local minima after a few more iterations. We will closely study this behavior, but nevertheless we anticipate that performing a more focused, greedier search will pay off in large domains such as the GPSS.

4 Summary

In this paper we put forth several enhancements to the current version of the Ground Processing Scheduling System (GPSS). Such enhancements will significantly improve its efficiency and usability by human schedulers. The enhancement is done by improving currently used algorithms and heuristics and by adding the extra feature of *fencing*. The fencing capability adds the flexibility of obtaining an optimal sub-schedule within a specified period of time.

References

[1] M. Deale, M. Yvanovich, D. Schnitzius, D. Kautz, M. Carpenter, M. Zweben, G. Davis, and B. Daun, 1994. The Space Shuttle Ground Processing Scheduling System. *Intelligent Scheduling*, edited by M. Zweben and M. Fox, *Morgan Kaufmann Publisher*, pp. 423-449.

[2] E. Freuder, R. Dechter, M. Ginsberg, B. Selman, and E. Tsang, 1995. Systematic Versus Stochastic Constraint Satisfaction. *IJCAI-95 Proceedings*, vol. 2, pp. 2027-2032.

[3] S. Kirkpatrick, C. Gelatt, and M. Vecchi, 1983. Optimization by Simulated Annealing. *Science*, vol. 220 #4598.

[4] S. Minton, M. Johnston, A. Philips, and P. Laird, 1992. Minimizing conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence*, vol. 58, pp. 161-205.

[5] P. Morris, 1993. The breakout method for escaping from local minima. *Proceedings of AAAI-93*, pp. 40-45.

[6] P. Selman and H. Kautz, 1993. An Empirical Study of Greedy Local Search for Satisfiability Testing. *Proceedings of AAAI-93*, pp. 46-51.

[7] P. Selman, H. Kautz, and B. Cohen, 1994. Noise Strategies for Improving Local Search. *Proceedings of AAAI-94*, pp. 337-343.

[8] G. Verfaillie, and T. Schiex, 1994. Solution Reuse in Dynamic Constraint Satisfaction Problems. *Proceedings of AAAI-94*, pp. 307-312.

[9] D. Waltz, 1975. Understanding line drawings of scenes with shading. *The Psychology of Computer Vision*, edited by P. Winston, McGraw-Hill.

[10] M. Zweben, E.Davis, B.Daun, and M.Deale, 1992. Rescheduling with iterative repair. *Proc. of AAAI 92* Spring Symposium on Practical Approaches to Scheduling and Planning.

[11] M. Zweben, E. Davis, and M. Deale, 1993. Iterative repair for scheduling and rescheduling. *IEEE Systems, Man, and Cybernetics*. Special issue on Planning, Scheduling, and Control.