Empirical Results on Scheduling and Dynamic Backtracking

Mark S. Boddy and Robert P. Goldman
Honeywell Technology Center, MN65-2200
3660 Technology Drive
Minneapolis, MN 55418
612-951-7403 612-951-7436 (FAX)
{boddy,goldman}@src.honeywell.com

KEY WORDS AND PHRASES

Constraint Satisfaction, Scheduling, Dynamic Backtracking

INTRODUCTION

At the Honeywell Technology Center (HTC), we have been working on a scheduling problem related to commercial avionics. This application is large, complex, and hard to solve. To be a little more concrete: "large" means almost 20,000 activities, "complex" means several activity types, periodic behavior, and assorted types of temporal constraints, and "hard to solve" means that we have been unable to eliminate backtracking through the use of search heuristics. At this point, we can generate solutions, where solutions exist, or report failure and sometimes why the system failed. To the best of our knowledge, this is among the largest and most complex scheduling problems to have been solved as a constraint satisfaction problem, at least that has appeared in the published literature.

This abstract is a preliminary report on what we have done and how. In the next section, we present our approach to treating scheduling as a constraint satisfaction problem. The following sections present the application in more detail and describe how we solve scheduling problems in the application domain. The implemented system makes use of Ginsberg's Dynamic Backtracking algorithm [2], with some minor extensions to improve its utility for scheduling. We describe those extensions and the performance of the resulting system. The paper concludes with some general remarks, open questions and plans for future work.

CONSTRAINT ENVELOPE SCHEDULING

We are interested in the solution of large, complex scheduling problems. A "solution" as we use the term is not simply an implementation of an algorithm for solving a particular constraint satisfaction or constrained optimization problem. For many domains, constructing schedules is an extended, iterated process that may involve negotiation among competing agents or organizations, scheduling choices made for reasons not easily implementable in an automatic scheduler, and last-minute changes when events do not go as expected. In such an environment, the process by which a schedule is constructed must be considered in any attempt to provide a useful scheduler for a given domain.

In our approach, which we call *constraint envelope scheduling*, schedules are constructed by a process of "iterative refinement," in which scheduling decisions correspond to constraining an activity either with respect to another activity or with respect to some timeline. The schedule becomes more detailed as activities and constraints are added. Undoing a scheduling decision means removing a constraint, not removing an activity from a specified place on the timeline.

The assumptions underlying our scheduling work are as follows:

1. Explicitly modelling the constraints resulting from specific scheduling decisions makes the schedule easier to construct and modify.

2. Representing only those relationships required by the current set of constraints (the decisions made so far) provides a more useful picture of the current state of the scheduling effort.

The main consequence of this approach is that the scheduler does not manipulate totally-ordered timelines of activities and resource utilization. Instead, the evolving schedule consists of a partially ordered set of activities, becoming increasing ordered as additional constraints are added (or less so, as those decisions are rescinded). This approach is common to a number of scheduling systems, e.g., [1, 5, 4, 3]

Figure 1 depicts the process by which a partially ordered schedule is gradually refined into an executable, totally ordered schedule. Although providing increased flexibility (through delaying commitment), the explicit representation of partially-ordered activities in the time map makes reasoning about resource usage and other state changes more complicated. It is no longer possible to construct a single time-line representing (e.g.) changing resource availability over time. Instead, the system computes *bounds* on the system's behavior.

Despite the approximate nature of this reasoning, we are still ahead of the game: where the least-commitment approach to scheduling can at least provide approximate answers in support of scheduling decisions (e.g. what order activities should occur in), timeline schedulers make the same decisions arbitrarily—putting an activity on the timeline is a stronger commitment than constraining it to occur (say) between two other activities, or within a given time window.

STATIC SCHEDULING FOR AVIONICS

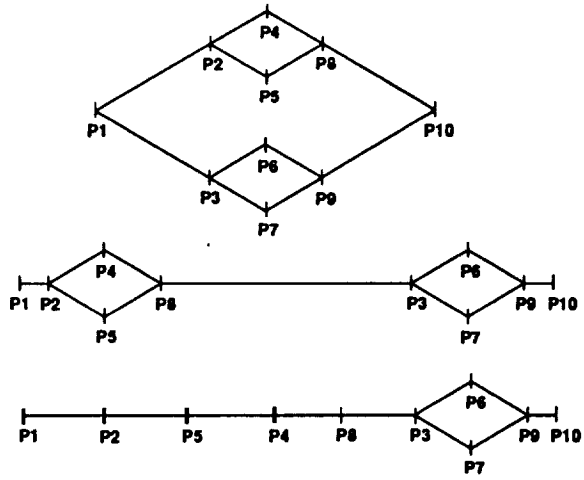

Figure 1: Gradual hardening of a partial order

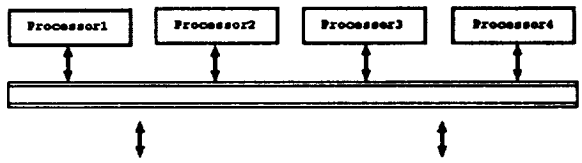

Figure 2: System architecture

One of the applications to which we have applied constraint envelope scheduling is static scheduling of processing time and bus communications in a distributed environment. This application involves safety-critical applications running on flight hardware on a commercial airplane. Figure 2 is a simple diagram of the architecture involved. The arrows at the bottom of the picture indicate that communication also occurs into and out of the cabinet in which the bus and processors reside. The schedule is static for reasons having to do with verifiability and repeatability of behavior, and ultimately with FAA certification for flight safety.

As we have already suggested, this problem is both large and complex. In a typical problem instance, there are approximately 6000 activities representing slices of processor time, and 14000 activites representing the transmission of data messages on the bus. There are six processors, which are between 80% and 90% loaded. The processes running on these pro-

cessors are periodic at rates between 5 Hz and 80 Hz. This makes the problem more complicated, in that data communication is specified between processes, not between process instances. One of the decisions to be made in constructing a schedule is to determine the mapping from instances of data producers to instances of data consumers. To make matters worse, we are constructing a schedule for a 200 mS "frame" which itself runs at 5 Hz. Communication from one instance of this frame to the next is entirely legal, and so we have in some sense a circular model of time, in which constraints on activites late in the frame may affect activities early in the frame.

Processes are to a limited extent preemptible, with minimum slice times and context-dependent context-switch times (i.e., it matters who you were preempted by). Interprocess constraints include jitter (bounds on how far from perfectly periodic instances of a process may be) and latency (limits on the time between producer and consumer instances for a given data message). There are data cycles, where process A gives a message to process B gives a message to process C, which sends a message back to process A. The interaction of these cycles with latency and jitter has complex effects on schedule feasibility. In fact, much of the work that we have done on this application has been the definition and derivation of conditions under which a given set of constraints was or was not consistent.

SCHEDULING AND DYNAMIC BACKTRACKING

The scheduler we have applied to this problem uses Ginsberg's Dynamic Backtracking algorithm [2], with some minor extensions. One of these extensions was to enable the search engine to report the set of inconsistent variables involved, should it fail to find a solution. For this application, knowing what constraints are in conflict is crucial: it enables us to go back to the system designers and tell them that their requirements cannot be met.

The second extension that we made was necessitated by the nature of the scheduling problem, or at least of how we have represented it. Ginsberg's algorithm involves generating *eliminations*: explanations of why a given value for some variable is ruled out given the current partial assignment. The assumption that eliminations are available by inspection does not work for complex temporal constraints: frequently we discover that a given ordering is infeasible by trying it. Accordingly, we have extended the algorithm to handle unsuccessful attempts to assign a given value to a variable. In this case, the search engine undoes the assignment (including removing any added constraints), records an elimination explanation for that value, and reports failure back to the scheduler.

Empirically, this extended implementation of Ginsberg's algorithm has been invaluable. A typical scheduling problem involves some tens of thousands of variables representing choices on ordering, preemption or producer/consumer pairing. Given the difficulty of localizing variable interaction, sorting related variables to be close to each other is impractical or impossible. Despite considerable effort, we have not managed to find variable or value ordering heuristics that result in backtrack-free solutions (we are currently using a variant of Smith's "slack" heuristic for value ordering [6]).

For these reasons, having a search method that leaves intact that part of a partial assignment not involved in a given inconsistency is crucial. One of the ways in which we might have run into trouble using dynamic backtracking has not materialized, either: inconsistencies typically involve less than 30 variables. This means that the elimination bookkeeping is kept within bounds, as well.

There is one feature of the current algorithm which has been inconvenient, however. The requirement that it be the most recently assigned variable that is re-assigned first clashes with the fact that in scheduling

applications there are frequently qualitative differences between variable types. For example, changing the ordering of an activity with respect to other activities using the same variable is in some sense a more local change to the schedule than changing the resource assigned to that activity. In the latter case, the activity must be ordered with respect to a different set of activities (those using the new resource). Any orderings remaining from the old resource assignment may now be for no purpose. For these reasons, we might like more flexible choices about variable ordering when backtracking.

CONCLUSIONS

The bottom line for this project is that we have had a successful impact on the solution of a hard problem that is a critical part of a multi-billion dollar investment. In the process of solving that problem, we have provided some empirical evidence that dynamic backtracking, suitably modified, is useful for nontrivial scheduling problems. We have also gained some useful experience in how to exploit the structure of the problem: heuristics are still critical to generating solutions or finding failures in a reasonable amount of time. "Reasonable" for this application currently means a small number of hours. Minutes would be better, days would be unworkable.

There is a lot of work yet to be done on this problem. For example, the problem is currently being solved in phases, with processor schedules being generated before the bus schedule. There are indications that heuristic repair techniques as in [7] might be useful for data scheduling.

One of the things we are hoping to arrange in the next few months is to release an instance or instances of this scheduling problem to the research community. Generation or accumulation of standard scheduling problems has been difficult. This problem has the advantages of being fairly challenging in both scale and complexity, and of having its roots in a real application.

References

[1] Fox, M.S. and Smith, S.F., ISIS: A Knowledge-Based System for Factory Scheduling, *Expert Systems*, 1(1) (1984) 25–49.

[2] Ginsberg, Matthew L., Interpreting Probabilistic Reasoning, *Proceedings of the 1985 AAAI/IEEE Sponsored Workshop on Uncertainty and Probability in Artificial Intelligence*, 1985.

[3] Muscettola, N., *HSTS: Integrating Planning and Scheduling*, Technical Report CMU-RI-TR-93-05, The Robotics Institute, Carnegie Mellon University, 1993.

[4] Sadeh, N. and Fox, M.S., Variable and Value Ordering Heuristics for Activity-based Jobshop Scheduling, *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management, Hilton Head Island, S.C.*, 1990.

[5] Smith, S.F., Ow, P.S., Potvin, J.Y., Muscettola, N., , and Matthys, D., An Integrated Framework for Generating and Revising Factory Schedules, *Journal of the Operational Research Society*, 41(6) (1990) 539–552.

[6] Smith, Stephen F. and Cheng, Cheng-Chung, Slack-Based Heuristics for Constraint Satisfaction Scheduling, *Proceedings AAAI-93, Washington, DC*, AAAI, 1993, 139–144.

[7] Zweben, M., Deale, M., and Gargan, R., Anytime Rescheduling, *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control, San Diego*, DARPA, 1990.