A Discipline for Reactive Rescheduling

J.E. Spragg
Robotics Institute, Carnegie Mellon University
Pittsburgh, PA 15213-3890, U.S.A.
jspragg@cs.cmu.edu

Gerry Kelleher
School of Computing and Mathematical Sciences
John Moores University, Liverpool
L3 3AF
U.K.
g.kelleher@livjm.ac.uk

Abstract

We present here a discipline for job shop rescheduling based on partial order backtracking. We show that partial order backtracking offers the rescheduler a framework for schedule repair, based upon a set of *nogoods*, which impose a systematic partial order on the set of activities to be repaired but allows non systematic techniques to be used within that framework. We present rescheduling as partial order backtracking by explaining the need for rescheduling, describing our framework (using a generic example), and discussing the particular implementation problems associated with repairing job shop schedules which consist of multi domain types.

Introduction

In a recent paper Ginsberg & McAllester (1994) suggested a hybrid search algorithm that combined the advantages of both systematic and non systematic methods of solving constraint satisfaction problems. The systematic search method of constraint satisfaction described by Ginsberg and McAllester, dynamic backtracking, employs a polynomial amount of justification information to guide problem solving. The non systematic methods, GSAT (Selman, Levesque, & Mitchell 1992) and min conflict (Minton et al. 1990) offer the search algorithm freedom to explore the search space by abandoning the notion of extending a partial solution to a CSP and instead modelling the search space as a total, if inconsistent, assignment of values to variables. A hill-climbing procedure is employed on this total set of assignments to try and minimize the number of constraints violated by the overall solution. Ginsberg and McAllester have called their hybrid algorithm *partial order backtracking.*

It is this hybrid algorithm which has been adapted by the current authors for rescheduling activities in a job shop environment. As Smith has noted (Smith 1995) in most practical environments, scheduling is an ongoing reactive process where evolving and changing circumstances continually force reconsideration and revision on existing schedules. Our aim in this paper is to describe partial order backtracking as a rescheduling procedure. To achieve this we will in section 1 outline the original algorithm as presented by Ginsberg and McAllester (1994). The next section will introduce our adaptation of the procedure which allows partial order backtracking to address the job shop rescheduling problem. The final section will suggest ways of implementing the algorithm on *real* rescheduling problems.

Partial Order Backtracking

Partial order backtracking brings a systematic search discipline to non systematic search procedures, such as GSAT and min conflicts repair, by applying the dynamic backtracking procedure developed by Ginsberg (Ginsberg 1993) to the search space.

Dynamic Backtracking

Dynamic backtracking maintains search information by accumulating a set of *nogoods*. A nogood is an expression of the form:

$$(x_1 = v_1) \wedge \ldots \wedge (x_k = v_k) \longrightarrow x \neq v$$

Here, a nogood is used to represent a constraint as an implication which is logically equivalent to the expression:

$$\neg[(x_1 = v_1) \wedge \ldots \wedge (x_k = v_k) \wedge (x = v)]$$

A special nogood is the empty nogood, which is tautologically false. If a empty nogood can be derived from the a given set of constraints, it follows that no solution exists for the problem being attempted.

New nogoods are derived by resolving old ones. As an example, suppose we have derived the following:

$$(x = a) \wedge (y = b) \longrightarrow u \neq v_1$$
$$(x = a) \wedge (z = c) \longrightarrow u \neq v_2$$
$$(y = b) \longrightarrow u \neq v_3$$

where v_1, v_2, and v_3 are the only values in the domain of u. Nogoods are combined to conclude that there are no solution with:

$$(x = a) \wedge (y = b) \wedge (z = c)$$

moving z to the conclusion of the above gives:

$$(x = a) \wedge (y = b) \longrightarrow z \neq c$$

The usual problem with maintaining a set of nogoods is that the set grows monotonically, at each step in the search a new nogood is added to the list of nogoods. Dynamic backtracking deals with this by discarding those nogoods whose antecedents no longer match the partial solution being extended by the search. Whilst this and related problems is being addressed by the RMS community (Tatar 1994, Kelleher & van der Gaag 1993) the approach of discarding contextually irrelevant nogoods appears to be a good strategy within the problems described by this paper.

Dynamic backtracking uses a set of nogoods to both record information about the portion of the search space that has been eliminated and to record the current partial assignment being considered by the procedure. The current partial assignment is encoded in the antecedents of the current set of nogoods. The antecedents of any set of nogoods, α, represent a consistent, if partial, solution to a constraint satisfaction problem (CSP). The next assignment must be an extension of this partial assignment. Assignments which have caused dead ends in the search can be detected by analyzing the conclusion parts of the nogood set. The dynamic backtracking, like most systematic search, assumes a static variable ordering. Whenever a nogood is added to the set of nogoods, the static variable ordering determines the variable that appears in the conclusion of the nogood. The most recently tried variable is *always* selected to appear in the conclusion of the new nogood.

POB

Partial order backtracking (POB) replaces the fixed variable ordering which constrains dynamic backtracking with a *partial* order that is *dynamically* sorted during the search. When a new nogood is added to the nogood set, this partial ordering does not fix a static sequence on the choice of variable to appear in the nogoods conclusion. As it turns out, there is considerable freedom as to the choice of the variable whose value is to be changed during backtracking, thereby allowing greater control in the directions that the procedure takes in exploring the search space.

However, there is not total freedom: ***safety conditions*** need to be maintained that model the partial ordering of the variables. It is necessary for variables in the antecedent of nogoods to precede the variables in their conclusion. This is because the antecedent variables are responsible for determining the current domains of such variables.

Rescheduling

Academic contributions to the scheduling literature have generally ignored the reactive view of problem solving and instead focused on the optimization of scheduling algorithms based upon a set of idealized problems which assume environmental stability. In this paper we present a reactive rescheduling discipline which approaches scheduling as a problem of *repair* over time. The emphasize of such a perspective argues for *operational* interests to take center stage: how fast is the scheduling system at responding to environmental change? How similar is the new version of the schedule to the old?

From a CSP point of view rescheduling introduces an extra set of constraints which need to be addressed. These are related to the need to preserve the old schedule as much as possible. The old schedule represents an investment in planned resources, allocation of machines and people, which should not be disturbed any more than necessary.

Example

A small rostering problem will demonstrate rescheduling as partial order backtracking. An airline has assigned teams of cabin staff to cover 5 flights over the next week. The flights are denoted by *flight1*, *flight2*, *flight3*, *flight4*, and *flight5* and the cabin staff by *red* crew, *yellow* crew, and *blue* crew.

Operational constraints demand that the same crews cannot work the following flight combinations:

flight1 and *flight2*
flight2 and *flight3*
flight2 and *flight4*
flight3 and *flight4*
flight3 and *flight5*
flight4 and *flight5*

A possible solution to this problem, supported by the following nogoods:

$$flight1 = blue \longrightarrow flight2 \neq blue$$
$$flight2 = red \longrightarrow flight3 \neq red$$
$$flight2 = red \longrightarrow flight4 \neq red$$
$$flight3 = yellow \longrightarrow flight5 \neq yellow$$
$$flight3 = yellow \longrightarrow flight4 \neq yellow$$
$$flight5 = red \longrightarrow flight4 \neq red$$

is

$flight1 = blue$
$flight2 = red$
$flight3 = yellow$
$flight4 = blue$
$flight5 = red$

Rescheduling will force a change of assignment on one of the variables already instantiated to a value. Let us assume that, for some operational reason, *flight3* must be covered by the *red* crew, what is the significance of this to the existing schedule? and what are the procedures necessary to discover this significance?

First, we must remove $flight3 = yellow$ from the set of nogoods. We also need to post safety-conditions with the set of nogoods that identify those existing assignments that are now suspect because *flight3*'s previous assignment determined their *live* domains at the time of their instantiation. These variables are recorded by the nogoods. In our example, *flight4* and *flight5* appear in the conclusion of those nogoods in which *flight3* is the antecedent.

$flight1 = blue \longrightarrow flight2 \neq blue$
$flight2 = red \longrightarrow flight3 \neq red$
$flight2 = red \longrightarrow flight4 \neq red$
$flight3 < flight4$
$flight3 < flight5$
$flight5 = red \longrightarrow flight4 \neq red$
$flight4 = blue \longrightarrow flight3 \neq blue$

The safety-conditions trigger consistency checks on the reordered variables *flight4* and *flight5*. We need to know what affect reassigning *flight3* will have on its *future* variables, those variables whose instantiations were determined after *flight3* was assigned. Consistency checks show that $flight5 = red$ is no longer consistent with the reassignment of *flight3*. *Flight5* is therefore removed from the set of assigned variables and appropriate nogoods posted.

$flight1 = blue \longrightarrow flight2 \neq blue$
$flight2 = red \longrightarrow flight3 \neq red$
$flight2 = red \longrightarrow flight4 \neq red$
$flight3 = red \longrightarrow flight4 \neq red$
$flight3 = red \longrightarrow flight5 \neq red$
$flight4 = blue \longrightarrow flight5 \neq blue$

However, we still have problems: we have to find an assignment for *flight5* that is consistent with the reassignment of *flight3*, and we need to make consistent those assignments that were made before *flight3* was assigned its air crew; that is, we have to make consistent *flight3*'s past variables, whose nogood conclusions forbid *flight3* from being assigned the *red* value.

We know from the nogoods that *flight5* cannot be assigned to either the red crew or the blue crew, but we can assign *flight5* the yellow crew. This we do:

$$flight5 = yellow$$

From the set of nogoods we know that $flight2 = red$ is inconsistent with $flight3 = red$. We therefore need to dynamically backjump to $flight2 = red$ and remove it from the assigned variable set. This we do, removing all inconsistent nogoods that were posted after *flight2*'s instantiation and posting new nogoods that identify *flight2*'s current *live* domain.

$flight1 = blue \longrightarrow flight2 \neq blue$
$flight3 = red \longrightarrow flight4 \neq red$
$flight3 = red \longrightarrow flight5 \neq red$
$flight3 = red \longrightarrow flight2 \neq red$
$flight4 = blue \longrightarrow flight5 \neq blue$
$flight4 = blue \longrightarrow flight2 \neq blue$

From the set of nogoods we know that the only consistent value that *flight2* can be assigned is yellow. We now have a consistent solution.

Rescheduling with POB Rescheduling differs from scheduling in a number of respects which are crucial for partial order backtracking as a rescheduling technique.

At least one extra constraint is introduced when rescheduling is initiated. This extra constraint encodes the *forced* assignment demanded by the rescheduling procedure. The constraint also imposes an order of repair on the existing schedule, splitting the pre-rescheduled solution into two new sub-problems: the problem of making consistent those instantiations which were made *prior* to the instantiation of the rescheduled variable, and the problem of making consistent those variables scheduled *after* the rescheduled variable. The new solution must, literally, be rescheduled around the changed variable. The first sub-problem is solved by 'squeezing' out the inconsistent assignments, moving the changed variable deeper into the search space if necessary, and the other by dynamically backjumping to assert a new ordering on the sequence of variables. Therefore, the POD procedure described by Ginsberg and McAllester must be modified to accept a consistent and complete solution, supported by a set of nogoods, and a new constraint which encodes a reassignment.

Implementation Issues

While our exposition of rescheduling as partial order backtracking gives the flavor of the procedure, job shop rescheduling presents a challenge which our simple pedagogical example was denied.

Implementation issues relating to the size and complexity of the problem are relevant here. The example given above takes as input a set of nogoods which support the complete and consistent solution. This is not practical once the number of variables is beyond a trivial amount. In such cases, relevant nogoods would need to be generated from the given constraints of the CSP on the fly. This has a processing time cost.

Again, our simple pedagogical example skipped over the question of how the rescheduled variable's future variables, those that appear in the conclusions of the nogoods of which it is antecedent, are to be repaired. This is an open question; there is no one answer. In the extreme case, they don't need to be repaired. The new assignment of the changed variable is consistent with all its future variable's instantiations. In other cases, arc consistency or heuristic repair is required. Which is the most appropriate depends upon the domain.

Another possibility might occur here, what if the rescheduled variable's future variables cannot be made consistent with the forced assignment? Intuitively what is required is to push the rescheduled variable deeper into the search space to introduce more variables into the pool of relaxed variables for rescheduling. However, this assumes that the reassignment of the changed variable is an hard constraint which cannot itself be relaxed. Assuming that it is, then the possibilities are that we chronologically backtrack into the rescheduled variable's past variables, or dynamically backtrack, preserving the instantiated variables between the rescheduled variable and the backtrack point. Dynamically backtracking would require the generation of further nogoods to locate the inconsistent variable's set of conflict variables.

Job shop Rescheduling

So far we have discussed rescheduling only over an idealized discrete domain with binary constraints. Unfortunately, job shop operations can rarely be described within such a neatly structured framework. The problem is in applying the forward checking procedure of constraint solving to multi-dimensional variables which range over a number of domains.

By multi-dimensional variables we mean those variables which are constructed when a one-to-many mapping is enforced between a job shop operation and (at least) two variables which represent the operation as *time* and a *resource*. Associated with each variable is a domain. The variable domains are the Cartesian product of the set of possible resources for an operation with the set of possible times for that operation.

The introduction of multi-dimensional domain types for each operation means that consistency checks, when the search procedure makes a tentative assignment of a value to a variable, must be performed both forward over the assigned value domains and backwards over other dependent domains. This complicates the construction of a nogood label somewhat!

Multi Domain Types There are a number of standard ways (Cheng-Chung & Smith 1995, Hasle, Kelleher, & Spragg 1995, Prosser 1993, Wiig 1995) in which scheduling problems can be encoded as a CSP. The scheduling world consists of jobs and operations on jobs. Let us assume that we have the following job set as part of a larger scheduling problem. We have two jobs, job_1 and job_2, where job_1 is composed of the operations $op_{1,1}$, $op_{1,2}$, and job_2 is composed of the operations $op_{2,1}$, $op_{2,2}$. Associated with job job_i will be an earliest start time es_i, and due date dd_i. An operation $op_{i,j}$ will have a demand for a resource and we may have a set of possible resources that will satisfy that requirement name $R_{i,j}$. In addition, the processing time of an operation may be dependent upon the resource used to perform that operation. That is, if we select resource $r_{i,j}$ for operation $op_{i,j}$, where $r_{i,j} \in R_{i,j}$, we can expect a processing time of $pr_{i,j}$. Resources of course have a limited capacity while operations are subject to technical constraints such as processing ordering, etc.

The simplest rendering of a scheduling problem as a CSP is to map scheduling operations onto CSP variables. However, such a rendering would produce variables with a number of domain types: resource domain, start time domain, inventory domain, etc. A better solution would be to represent each scheduling operation as a number of variables which range over their own domain types so that we have a one-to-many mapping between an operation and CSP variables of the types, start-time, resource, etc. However, this multi-dimensional CSP representation of the scheduling world presents some problems for both constraint solving and the construction of nogood labels.

The job shop scheduling problem is to allocate an operation to an interval in time on a resource such that all constraints, including optimization constraints (which we will ignore here) are satisfied. The CSP mapping of the scheduling world consists of a set of variables of domain type *resource*, $\{v.r_{1,1}, v.r_{1,2}, v.r_{2,1}, v.r_{2,2}\}$, and a set of variables of domain type *time*, $\{v.t_{1,1}, v.t_{1,2}, v.t_{2,1}, v.t_{2,2}\}$. The domains of the *resource* variables are $[r_1, r_2]$, and the domains of the *time* variables are $[0 \,.\, 100]$. The resource r_1 has a processing time of 3, while r_2 has a processing time of 2. Both resources have a capacity of 1.

Immediately we see the problem of employing multi-dimensional domain types. Imagine we wish to make a tentative assignment of, say, r_1 to variable $v.r_{1,1}$, we need to check both backwards over the *time* domain to see for what period the resource is available and forwards over the *resource* domain to see if the resource can be removed from any future domains. At this stage our nogood labelling gets a bit more complicated but becomes essential for constraint maintenance. Again imagine that we assigned r_1 to variable $v.r_{1,1}$ for the period $[0 \,.\, 3]$, a typical nogood label would be:

$$v.r_{1,1} = r_1 \land v.t_{1,1} = [0 \,.\, 3]$$
$$\longrightarrow$$
$$v.r_{2,1} = r_1 \land v.t_{2,1} \neq [0 \,.\, 3]$$

Which is to say that variable $v.r_{2,1}$ could be assigned r_1 at any other time other than between $[0 \,.\, 3]$. The *conclusion* of the nogood has become a conditional. This has major implications for nogood resolution, and, indeed, dissolution during rescheduling. For example, if variable $v.r_{1,2}$ was later assigned r_1 over the period [3 . 6] then our nogood would become:

$$v.r_{1,1} = r_1 \wedge v.r_{1,2} = r_1 \wedge v.t_{1,1} = [0\ .\ 3] \wedge v.t_{1,2} = [3\ .\ 6] \longrightarrow v.r_{2,1} = r_1 \wedge v.t_{2,1} \neq [0\ .\ 6]$$

What is the significance of multi domain types to partial order backtracking and rescheduling?

Partial Order Backtracking Over Multi Domain Types

In the job shop rescheduling problem, new jobs can be added to the set of jobs to be scheduled, jobs can be removed, resources can be reassigned. The encoding of nogood conclusions as conditionals furnishes for these actions sufficient information to allow a partial order to be maintained over all domains. For example, given our nogood's conditional conclusion:

$$v.r_{2,1} = r_1 \vee v.t_{2,1} \neq [0\ .\ 6]$$

we know that the variable $v.r_{2,1}$ cannot be assigned to r_1 during the period [0 . 6] because of the past choices encoded in the nogood's antecedent:

$$v.r_{1,1} = r_1 \wedge v.r_{1,2} = r_1 \wedge v.t_{1,1} = [0\ .\ 3] \wedge v.t_{1,2} = [3\ .\ 6].$$

What is true of variable $v.r_{2,1}$ is also true of any new variable derived from the introduction of an additional operation into the schedule. That is, if the schedule user wishes to assign the resource r_1 to a new variable $v.r_{x,y}$ after 0 but before 6 it cannot do so because of the same reasons that variable $v.r_{2,1}$ cannot be assigned to that resource during that period. The nogood must be copied, substituting $v.r_{x,y}$ for $v.r_{2,1}$, allowing $v.r_{x,y}$ to inherit $v.r_{2,1}$'s antecedent information. It is this information which is used to repair the schedule.

Again, the set of nogoods allows consistency to be maintained once a job has been removed from the schedule. The removal of variable $v.r_{1,2}$ for example will allow our nogood's conclusion to be rewritten has:

$$v.r_{2,1} = r_1 \vee v.t_{2,1} \neq [0\ .\ 3].$$

The reassignment of a resource is equivalent to the generic case described above. However, because of the multi-dimensional search space, additional arc consistency needs to be maintained over temporal domains representing the start time and duration of an operation.

Conclusions

We have presented a discipline for job shop rescheduling based on partial order backtracking. We have shown that partial order backtracking offers the rescheduler a framework for schedule repair based upon a set of nogoods which impose a systematic partial order on the set of activities to be repaired. We have also shown that by representing the conclusion of a nogood as a conditional statement we can also impose a partial order between domains; we can determine the availability of a particular resource by representing its dependency on its temporal domain.

An interesting point to note about this approach is that, as a consequence of it's basis in POB, it makes what may be called a "strong single context assumption" about the maintenance of dependency information in supporting rescheduling. What is meant by this is that, as POB effectively throws away dependency information about everything other than the current assumptive context (the assumptions made for the particular current solution or partial solution being considered) it can only ever consider information from the single context within which it is operating. The motivation for this in Ginsberg's work is the desire to avoid the overhead in memory of maintaining unwanted dependencies. One might argue about the tradeoffs between systems that maintain such information and provide other benefits - such as the emerging generation of focussed RMS - but the fact remains that in embracing POB one is accepting the efficacy of a single context in supporting dependency based reasoning.

Making this strong single context assumption has an interesting side-effect when rescheduling is considered. The rescheduling approach described here is independent of the evolution of the schedule, we are not reasoning with dependencies about how we reached a schedule only with those dependencies relating to the development of a particular schedule (the current one). As a consequence we may use the approach to provide rescheduling capability to any scheduling system, irrespective of the way in which the system operates. We can add on rescheduling to any system straightforwardly and efficiently provided some simple information is available to us, namely a set of constraints and variables describing the initial problem and the schedule being modified. The idea is that we realize the solution in the CSP described by the initial problem state maintaining the dependency information as we go. From this information we may perform rescheduling as described in the body of this paper. Further detail on the approach and related work may be found in Kelleher & Spragg (1996), Hasle, Kelleher, & Spragg (1995) and Spragg & Kelleher (1995).

Acknowledgments.

This work was funded by the TRUTH Project; European Communities ESPRIT III Programme, Project Number: 6463. We gratefully acknowledge the assistance of Geir Hasle, and Jonathan Cunningham.

References.

Cheng-Chung, C., and Smith, S.F. 1995. *Applying Constraint Satisfaction Techniques to Job-Shop Scheduling* The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, CMU-RI-TR9-95-03.

Ginsberg, M.L. 1993.
Dynamic Backtracking Journal of Artificial Intelligence Research 1, 25-46.

Ginsberg, M.L., and McAllester, D.A. 1994.
GSAT and Dynamic Backtracking Knowledge Representation and Reasoning Conference.

Hasle, G., Kelleher, G., and Spragg, J.E. 1995.
Encoding the Pirelli Tyre Scheduling Problem as a CSP Paper accepted for IJCAI-95 Workshop on Intelligent Manufacturing Systems.

Kelleher, G. and Spragg, J.E., 1996.
Add-On Rescheduling: Rescheduling with Arbitrary Scheduling Systems John Moores University, School of Computing and Mathematical Sciences, Report CMS 14.

Kelleher, G., and Van Der Gaag, L. 1993.
The LazyRMS: Avoiding Work in the ATMS Computational Intelligence. Vol. 9, Number 3.

Minton, S., and Johnston, M.D., Philips, A.B., and Laird, P. 1990.
Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In Proceedings of the Eighth National Conference on Artificial Intelligence, pages 17-24.

Prosser, P. 1993.
Scheduling as a Constraint Satisfaction Problem: Theory and Practice In Scheduling of Production Processes edited by Dorn and Froeschi. Ellis Horwood.

Selman, B., Levesque, H., and Mitchell, D. 1992.
A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440-446.

Smith, S.F. 1995.
Reactive Scheduling Systems. Center for Integrated Manufacturing Decision Systems, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213.

Spragg, J.E., and Kelleher, G. 1995.
Context Shifting with Constraint Satisfaction Problems: Dynamic Backtracking with Forward Checking John Moores University. School of Computing and Mathematical Sciences, Report CMS 8.

Tatar, M.M. 1994.
Combining Lazy Evaluation with Focusing Techniques in an ATMS Proceedings ECAI94.

Wiig, O.H. 1995.
Constraint Reasoning and Dynamic Scheduling SINTEF Report STF33 A95001.