A REACTIVE SCHEDULING AGENT

Patrick Prosser
University of Strathclyde
Department of Computer Science
26 Richmond Street
Glasgow GI 1XH, UK

ABSTRACT

Factory scheduling can be considered as an instance of the constraint satisfaction problem (CSP). Factory scheduling differs from traditional forms of the CSP in that it is a *dynamic* or *open* problem. Constraints are added to and retracted from the problem as work progresses on the shop floor. Constraints alter as unexpected events occur, such as the breakdown of machines, the late arrival of work, the early arrival of work and the changing demands put upon the scheduling system by the user. A system is required that can generate and maintain a schedule in near-real time and exploit opportunities as they arise. A single resource scheduling agent has been developed to schedule incrementally and reactively in a dynamic environment using advanced CSP techniques.

1. THE SCHEDULING PROBLEM

The scheduling task can be loosely described as the problem of assigning resources and start times to operations. One conventional approach is to view this problem as one of *optimisation.* Two problems are associated with such a strategy: *definition* and *brittleness.* In many real world applications the concept of *optimality* is poorly defined due to problems of measure and conflict. To optimise on one measure alone could jeapordise other measures of optimality. Due to the dynamic and stochastic nature of the factory environment, *optimal* schedules quickly break down. They are *brittle.* It is preferable to produce schedules that are *satisfactory* and to maintain these schedules in the dynamic environment.

Schedulers can be classified as *predictive* or *reactive.* Predictive schedules are created in a static world and assume that events are entirely predictable. Reactive scheduling addresses the problem of maintaining a schedule in a dynamic and stochastic world. Such a world offers up both conflicts and opportunities to the scheduler. In ISIS [Smith et al., 1986] deviations from the predictive schedule are viewed as constraint violations which result in a rescheduling of the effected orders.

In OPIS [Ow et al., 1988] and SONIA [Collinot et al., 1988] the predictive and reactive scheduling components are integrated. Reactive scheduling (schedule maintenance) is achieved by the use of domain specific heuristics: load balancing, constraint relaxation and permutations of the predictive schedule. The system described here creates incrementally a predictive schedule (in a fashion similar to that used by Elleby [1987]) and maintains that schedule, reacting to both conflict and opportunity, using one common mechanism.

2. PROBLEM REPRESENTATION

The scheduling problem can be represented using constraints. The job (or customer order) is expressed as a process plan, that is, a (possibly non-linear) sequence of job steps (as in [Fox and Kempf, 1985]) that must be performed to complete the job. The process plan is then the set of *precedence* constraints on a job. Each job step is refered to as an operation. Associated with each operation are *technological* and *temporal* constraints. Technological constraints describe the set of resources that can be used to perform a given operation. Temporal constraints describe the intervals of time in which an operation can be performed such that the due date of the order is met. In the single resource scheduling case it is assumed that the precedence and technological constraints have been satisfied; the problem is to satisfy temporal constraints on individual operations.

The CSP can be represented (as in [Mackworth, 1977]) as a simple un-directed graph G. Let V(G) the set of vertices in G correspond to variables {v_1, v_2, v_3,. . . .v„}. Each variable v, has a domain *d{.* a set of values that may be assigned to v,. Let E(G) be the set of binary constraints {(v, v;), (v_k v_1), ... (v_m vj}. between pairs of variables. The binary constraint (v, v_y) exists if there are values in the domain d_k that may conflict with values in the domain d_r

In the scheduling problem an operation *OP,* is analogous to a variable v,. The domain d_i is then the set of legal start times for OP_i. The binary constraint (OP, *OPj)* exists if there are values in d_i that may conflict with values in *dj.* Viewed another way there is a potential to schedule *OPi* and *OPj* such that they interfere in time. During the construction or execution of a schedule the domain *d,* of operation *OPi* may be altered by an external agent.

Domain d_i may be enlarged (OP_i may arrive early or be allowed to finish late) or reduced (OP_i, may arrive late or have to finish early). A new operation *OPj* can be added to G (an operation arrives at the resource) or an existing operation OP_k can be deleted from G (an operation completes and leaves the resource). These changes may create and destroy edges in G.

Most research on the constraint satisfaction problem has used the n-queens or similar problems as a vehicle to study algorithms and their complexities. The n-queens problem has two characteristics that put it apart from the factory scheduling problem. Firstly, the n-queens problem can be represented by a complete undirected graph G (sometimes refered to as K_n). In the factory scheduling problem the complete graph K_n will be a special case. The graph G will tend to be incomplete and may be a union of disconnected graphs, corresponding to a set of independent CSP's. Secondly, the n-queens problem is a *static* or *closed* constraint satisfaction problem: once posed the graph G does not alter. In the factory scheduling problem the graph G is *dynamic.*

One further difference between the traditional CSP and the one posed here is that only the first solution to the scheduling problem is required (though if the problem is one of optimisation then many schedules would be required). The search for the first solution in a dynamic problem has implications on the techniques to be used. A search appropriate to finding all solutions (as in Mack worth [19771) is not neccessarily economical in finding the first solution.

3. THE REACTIVE SCHEDULING AGENT

The agent is given the scheduling problem posed as a constraint graph G. In moving forwards through the search space the agent performs forward checking, removing inconsistencies from the domains of unscheduled operations. Moving backwards, backtracking, the agent reasons over inconsistencies deriving knowledge about the search space. This knowledge is exploited during search (pruning the search space) and in reacting to externally induced change. The agent exploits the topology of G in two ways [Nudel, 1983]. First, having scheduled operation *OP,* the agent will select the next operation to schedule from the set of unscheduled operations adjacent to OP_i. This allows the agent to navigate around independent problems rather than imbed them within each other. Secondly, having given *OP,* a start time forward checking need only be applied to the set of unscheduled operations adjacent to *OP,*. In moving backwards through the search space, backtracking, the agent exploits the dependency information derived by forward checking.

The following example demonstrates the incremental creation of a schedule by the agent. Suppose we have four operations (A, B, C and D) to be scheduled on a single resource. For the sake of simplicity it is assumed that each operation has a duration of 1 unit of time on the resource, the resource cannot be shared and the temporal domains of the operations are discrete (table 1).

OPERATION	DOMAIN
A	2 3
B	1 3 5
C	2 4
D	1 2 4

Table 1.

The scheduling problem is represented as a constraint network. In the example given an arc (constraint) exists between operations *OPi* and *OPj* if there is an intersection between their domains (a potential for operations to interfere in time). Table 2 shows the adjacency matrix for the constraint graph V(G)=(A B C D).

	A	B	C	D
A	0	1	1	1
B	1	0	0	1
C	1	0	0	1
D	1	1	1	0

Table 2.

The agent solves the scheduling problem in a depth-first incremental manner. On assigning a start time to an operation *OPi* the agent applies the forward checking procedure (FC) of Haralick [Haralick and Elliot, 1980] to all unscheduled operations adjacent to *OP,* in the constraint graph G, removing inconsistent values from their domains. Table 3 shows the result of assigning a start time of 2 to operation A. The value 2 is removed from A's domain. Forward checking is applied to all operations adjacent to operation A: operations B, C and D. This results in the reduction of the domains of operations C and D (the value 2 is removed from their domains) and a recording of the source of this reduction (column *reducers* in table 3).

OP	START	DOMAIN	REDUCERS
A	2	3	-
B	-	1 3 5	-
C	-	4	A
D	-	1 4	A

Table 3.

The agent will then proceed, making the assignments B=l (forward checking B with D) and C=4 (forward checking C with D). This results in, table 4, operation D having a null domain.

OP	START	DOMAIN	REDUCERS
A	2	3	-
B	1	3 5	-
C	4	-	A
D	-	Ø	C B A

Table 4.

On discovering an inconsistency the agent will backtrack using dependency information derived by forward checking. During backtracking the agent reasons over inconsistencies using the *nth order shallow learning* of Dechter [1986]. In the situation depicted in table 4 the agent will concentrate on the operation with the null domain, operation D, until the inconsistency is resolved or the information derived by forward checking is exhausted. The most recent reducer of D, operation C, will be unscheduled. This will enlarge the domain of D, returning the value 4 that was earlier removed by forward checking C with D. Using shallow reasoning the agent determines the nogood *{(D€V(G) & B=1 & A=2) -> C#4],* (when operation D exists and B=I and A=2, C cannot have the value 4). For brevity the nogood is identified uniquely by a symbol, in this case #0. Due to nogood #0 the value 4 is removed from the domain of C leaving C with a null domain. The agent will then unschedule the next most recent reducer of operation D, operation B. As a result of this nogood #0 can no longer be believed because B=I is false and the value 4 is returned to the domain of operation C (nogood #0 is destroyed). Unscheduling B will also return the value 1 to the domain of D, previously removed by forward checking B with D. Again using shallow learning the agent derives the nogood *#]{(A=2 & CeV(G) &. DeV(G)) —> B#1},* removing the value 1 from the domain of B leaving a domain of (3 5). This is shown in table 5.

OP	START	DOMAIN	REDUCERS
A	2	3	-
B	-	3 5	#1{(A=2 & C∈ V(G) & D∈ V(G)) → B≠1}
C	-	4	A
D	-	1 4	A

Table 5.

Search can then move forward from B, making the assignments B=3, C=4 (forward checking C with D) and finally D=I (table 6).

OP	START	DOMAIN	REDUCERS
A	2	3	-
B	3	5	#1{(A=2 & C∈ V(G) & D∈ V(G)) → B≠1}
C	4	-	A
D	1	-	CA

Table 6.

The nogood #1{(A=2 & C€V(G) & D€V(G)) -> B#1} is equivalent to the discovery of a *path inconsistency* between vertices A, B, C and D in the constraint graph G.

4. REACTION TO EXTERNALLY INDUCED CHANGE

Notification of change is received by an agent via mail. Typically changes are: the deletion of an operation, the addition of an operation or a modification to the domain of an operation. Such changes can occur during search or after search has completed. Deletion of an operation, *$0P_{x}$*, corresponds to the relaxation of constraints. All reductions imposed by *$0P_X$* via forward checking are returned to the domains of the effected operations, the nogood knowledge-base is updated and *OP_x* is removed from the constraint graph. For example suppose operation C is removed from the scheduling problem in table 6. The reductions imposed by C on D via forward checking are undone (the value 4 is returned to the domain of D). The nogood #7 *{A =2 & C€V(G) & D€V(G) -> B#1}* can no longer be believed because C€V(G) is false. The nogood #1 is destroyed and the value 1 is returned to the domain of B. Operation C is finally removed from the constraint graph resulting in the schedule of table 7.

OP	START	DOMAIN	REDUCERS
A	2	3	-
B	3	1 5	-
D	1	4	A

Table 7.

The addition of an operation *OP_x* corresponds to the addition of constraints to the problem and involves the mechanisms of search (forward checking, dependency directed backtracking and shallow learning). Suppose operation E, with a single point domain (2), is added to the schedule in table 6. The adjacency matrix (table 2) is updated, creating new arcs AE, EA, CE, EC, DE and ED. Forward checking takes place between A and E, removing the value 2 from E's domain, resulting in an inconsistency (table 8).

OP	START	DOMAIN	REDUCERS
A	2	3	-
B	3	5	#1{(A=2 & C∈ V(G) & D∈ V(G)) → B≠1}
C	4	-	A
D	1	-	C A
E	-	Ø	A

Table 8.

The agent will then perform the following sequence of actions, resulting in the schedule depicted in table 9.

```
1     UNSCHEDULE A
1.1     A#2 therefore relinquish
        #1{(A=2 & C€V(G) & De V(G)) -> B#1}
1.2     Return to domain of B the value 1
1.3     Undo forward checking from A.
        Return value 2 to domains of C, D and E
1.4     Create nogood #2{(E€V(G)) -> A#2}
1.5     Apply FC(BA) removing 3 from domain of A.
        Domain of A is 0
2     UNSCHEDULE B
2.1     Undo forward checking from B.
        Return value 3 to domain of A
2.2     Create nogood #3{#2 -> B#3}
2.3     Apply forward checking between D and B
        removing the value 1 from domain of B
3     MAKE ASSIGNMENTS
3.1     B←5
3.2     A←3
3.3     E←2
```

The nogood #2{(EeV(G)) —> A#2) is equivalent to discovery of an *arc inconsistency* between vertices E and A in the constraint graph G.

OP	START	DOMAIN	REDUCERS
C	4	2	-
D	1	2	C
B	5	-	D & #3(#2 $\rightarrow$ B$\neq$3}
A	3	-	#2{(E$\in$ V(G)) $\rightarrow$ A$\neq$2}
E	2	-	-

Table 9.

A naive method of reacting to a modified domain is to delete the effected operation and add it back into the problem as an unscheduled operation using the techniques described above. Such a technique makes little effort to protect the knowledge discovered during search and is not recommended. Assume the original domain D, of operation OP_x has been modified to become a new domain D_v The following situations must be considered:

(a) $D_{x_n} \subset D_{x_o}$
(b) $D_{x_n} \supset D_{x_o}$
(c) $\neg(D_{x_n} \subset D_{x_o}$ or $D_{x_n} \supset D_{x_o})$
&
(1) OP_x is scheduled and its start time falls within D_x.
(2) OP_x is scheduled and its start time falls outside D_x,
(3) OP_x is not scheduled.

In (a) the domain of OP_x has been reduced. Typically this is due to some earlier operation in the process plan of OP_x finishing late, some later operation in the process plan being re-scheduled into a earlier time interval or the due date on the order becoming more restrictive. In situation (b) the domain of OP_x has been enlarged. This may be due to some earlier operation in the process plan finishing earlier than anticipated, some later operation in the process plan being re-scheduled into a later time interval or a relaxation of the due date of the order. In (c) the domain of $0P_X$ has been translated along the time line and, possibly, enlarged or reduced. This can be caused by a potentially malignant combination of events from both (a) and (b).

Situations (a), (b) and (c) must be considered in conjunction with situation (1), (2) and (3), giving cases (a.l) through to (c.3). In fact there are only eight cases to consider. Case (b.2) is a contradiction and therefore cannot occur. Cases (a.2) and (c.2) correspond to conflicts: the start time of an operation now lies outwith its temporal domain. This can be addressed by dependency directed backtracking. Case (b.l) corresponds to an opportunity, the operation can be re-scheduled using a start time which better meets the order's due date or the resource's scheduling goals.

In situations (b) and (c) the domain of an operation has been enlarged or translated and possibly reduced/enlarged. Any nogood with OP_x€V(G) as part of its antecedent must be relinquished; the nogood can no longer be believed. For example assume, in table 9, the domain of operation E is enlarged from (2) to become (1 2). The nogood *#2{(EeV(G)) —> A#2}* can no longer be believed because we have not considered the implications of E having the additional value 1 in its domain. [In fact the value 2 in the domain of A is still nogood, although it has not yet been deduced, so long as *(C€ V(G) & De V(G) & E€ V(G))* remains true]. If we can no longer believe #2 then we can no longer believe #3{#2 —> *B#3}.* Therefore enlarging or shifting the domain of E also enlarges the domains of A and B. A further effect of enlarging the domain of an operation is the potential addition of arcs to E(G). In the case above, E's domain enlarged to (1 2), new arcs BE and EB must be added to E(G).

In situation (a) the domain of OP_x has become more restrictive and the nogoods in the knowledge-base continue to be believed. For example in table 6, if the domain of operation C is reduced from (2 4) to (4) we continue to believe *#1{(A=2 & CeV(G) & DeV(G)) -> B#1).* Reducing C's domain from (2 4) to (4) will remove the arcs AC and CA from E(G).

5. ENHANCEMENTS

The basic algorithm described in sections 3 and 4 uses heuristics to improve performance when applied to the scheduling problem.

The arcs in E(G) are classified as either *active* or *passive.* When forward checking is applied across an arc, with the consequent reduction of a domain, the arc is considered active. The next operation to be scheduled is selected from the set of unscheduled operations incident to active arcs. For example, in table 3, after operation A<— 2 the agent would make an assignment to C or to D and then finally to B without backtracking. A further improvement is made by *locally reordering consistency tests* [Nudel 1983]) after each call to forward checking that results in the discovery of an inconsistency.

Having determined the set of candidate operations to be considered for scheduling a heuristic is applied to select the most appropriate operation to schedule. The *fail-first* heuristic of Haralick [1980] (select the operation with the smallest domain) is not appropriate to the factory scheduling problem. The reason for this is that the domains and durations of operations in G are not all identical (as in the n-queens) and an operation with the smallest domain need not be the operation that most constrains the problem. The *greatest valency* heuristic of Nudel [1983] (select the vertex with maximum valency) has also performed relatively poorly in this domain. The best performer, so far, has been *greatest duration.* This heuristic selects from the set of candidates the operation with the longest duration. This is a similar heuristic to that used in bin packing, fitting large blocks first then smaller blocks in the surrounding gaps.

6. THE AGENTS ROLE

The agent described here is a component part of DAS, the Distributed Asynchronous Scheduler [Buchanan et al, 1988]. DAS is a reactive/opportunistic scheduler, currently undergoing field test at Alcan Plate Ltd (Birmingham, England). The architecture of DAS is a model of the ideal business organisation. DAS is a three level hierarchy of units. At the lowest, *operational,* level each unit represents an individual machine centre. At the middle, *tactical,* level each unit represents an aggregation of similar machine centres. The top, *strategic,* level represents a global management view of the scheduling task. Attached to each unit in the hierarchy is a scheduling agent. At the operational level *o~agents* maintain work-to-lists for individual machine centres. At the tactical level *t-agents* delegate and retract work from the subordinate operational level. At the strategic level the *s-agent* releases work onto the shop floor. All agents run asynchronously and communicate freely via mail (although agents are allowed to ignore messages under special circumstances). The o-agent's role is to satisfy temporal constraints. The t-agents satisfy technological constraints and attempt to resolve operational conflicts by load balancing between resources. The s-agent resolves temporal conflicts by inter-agent backtracking (resequencing the order that decisions are made through a process plan) or by temporal constraint relaxation.

The agent described here is the o-agent. Decisons made by an o-agent are propagated through the global hypothesis (the current schedule) and messages are sent to affected agents. Change induced on one operational agent by any another agent are handled in an identical manner to change induced on an agent from the external world. If the operational agent is posed an over constrained scheduling task the agent will deliver to its superior t-agent a conflict set (a set of operations that cannot be consistently scheduled on this resource). This conflict set is derived from the agent's current set of beliefs, the nogood set. The tactical agent analizes this conflict set and attempts to resolve it by load balancing (undoing technological decisions). If the t-agent fails to resolve a conflict then it will pass up a new, richer, conflict set to the s-agent. This may result in the s-agent resequencing the order that decisions are made through a job's process plan or as a last resort relaxation of temporal constraints on one of the jobs in the conflict set.

7. CONCLUSION

The reactive agent is a hybrid of forward checking, shallow learning and the JTMS of Doyle [1979]. The implementation differs from the pure JTMS in the following respects

- The current set of program beliefs and then-jus tifications is, exclusively, the set of nogoods.
- All justifications have empty out-lists's
- Any program belief that is not justified is deleted
- The incremental schedule is the set of program assumptions
- The creation of program beliefs is performed by shallow learning based on knowledge derived from forward checking
- Backtracking only uses dependency information derived from forward checking
- The set of program beliefs (the nogood set) are utilised in pruning the search space, in reacting to externally induced change and in explaining conflicts to the external world.

By giving an agent a reactive/opportunistic capability and an ability to explain its current set of beliefs (due to shallow learning and truth maintenance techniques) a symbiosis emerges. Problem solving effort can be distributed across an architecture and be allowed to proceed asynchronously.

REFERENCES

[Buchanan et al., 1988] Iain Buchanan, Peter Burke, John Costello and Patrick Prosser. "A Distributed Asynchronous Scheduler", Technical Report AISL36, Department of Computer Science, University of Strathclyde

[Collinot et al, 1988] Anne Collinot, Claude le Pape and

Gerard Pinoteau. "SONIA: A Knowledge-based Scheduling System", Artificial Intelligence in Engineering, vol 3, pp 86-94, 1988

[Dechter, 1986] Rina Dechter. "Learning While Searching In Constraint Satisfaction Problems", AAAI-86, vol. 1, pp. 178-183, August 1986.

[Elleby et al, 1987] Peter Elleby, Hugh Fargher and Tom Addis. "Incremental Constraint Satisfaction", Knowledge Systems Group, Department of Computer Science University of Reading, Whiteknights Park, Reading, Berkshire

[Doyle, 1979] Jon Doyle. "A Truth Maintenance System", Readings in Artificial Intelligence, pp 495-516

[Fox and Kempf, 1985] B.R.Fox and K.G. Kempf. "A Representation for Opportunistic Schedules", Robotics Research, The Third International Symposium, pp 109-115

[Haralick and Elliot, 1980] Robert M Haralick and Gordon L Elliot. "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", Artificial Intelligence, vol 14, pp. 263-314, 3/10/1980

[Mackworth, 1977) Allan K Mackworth. "Consistency in Networks of Relations", Artificial Intelligence, vol. 8, pp. 99-118, 1977

[Nudel, 1983| Bernard Nudel. "Consistent Labelling Problems and their Algorithms: Expected Complexities and Theory Based Heuristics", Artificial Intelligence, vol. 21, pp. 135-178, 1983

[Ow et al., 1988] Peng Si Ow, Stephen F Smith and Alfred Thirez. "Reactive Plan Revision", Proceedings of the CA1A-88, pp 77-82, 1988

[Smith et al.,1986] Stephen F Smith, Mark S. Fox, and Peng Si Ow. "Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems", AI Magazine, Fall 1986