Constraint Based Reactive Rescheduling in a Stochastic Environment.

J.E. Spragg
Mitthögskolan, Sweden.

G. Fozzard
De Montfort University, Leicester UK.

and

D. Tyler
Manchester Metropolitan University, Manchester, UK.

April 1997

Keywords

Schedule repair, flow lines, garment manufacture, partial order backtracking, reassignment heuristics, and dynamic CSP.

Abstract

The problem of scheduling manufacturing systems where the performance, or indeed, capacity of a production resource is subject to stochastic change, is the subject of this paper. Typical of such resources are those which are dependent upon labour intensive processes.

In the United Kingdom the manufacture of clothing garments is still dominated by the progressive bundle system (PBS).[1] Garments are produced on a continuous-flow production line in which garment pieces are passed in succession through a network of workstations where skilled manual workers complete operations on garments using sewing machines. The workstations which comprise the flow line can be connected in either a serial or a parallel fashion depending upon the sequencing constraints which govern the order in which the garment are to be assembled. The rate of flow of work through each workstation is determined by the performance of the machinists.

The dominance of the PBS relates to the difficulties of automating the assembly of garment pieces. The automation of handling a flimsy material like cloth alone has proved a major bottleneck in the adoption of flexible manufacturing practices. Despite advances in automation in other sectors of manufacturing industry, it is clear that the requirement to successfully manage skilled manual labor will continue to be employed until appropriate technologies can be found. The PBS represents a serious scheduling problem for factory managers and line supervisors. This problem is not alleviated by the trend towards smaller contract sizes which reflect a fluid fashion market.

The determination of the optimum order in which sewing operations should be arranged is not a serious scheduling problem. In fact, it is the classic sequencing problem, ***n/m/P/Cmax***, discussed in French (1982)[2]. The real scheduling problem associated with a PBS arises from the constant need for reactive rescheduling to maintain line balance. The frequency of operator absenteeism, or machine breakdown, or unstable operator performance, requires constant reassignment of operators and operations. In such systems, reactive rescheduling becomes so frequent that it takes on the character of supervisory control. In the approach described here, line balance is maintained via periodic schedule repair, based upon reassignment heuristics, supported by partial order backtracking.

Line Balance

Schedule repair activity is triggered by monitoring the flow line. At the start of a new line, operators are allocated to operations by line supervisors and production managers. The operators required for each operation is calculated using the principles of load and capacity planning. In practice, the operators' skills and performance rarely fit the work content of the operations and potential bottlenecks become inherent in the line's design. Moreover, even if it were possible to achieve a perfect line balance, it would be impossible to maintain it over time due to line perturbations caused by machine breakdowns, operator absenteeism and fluctuations in operator performance. The schedule must be repaired to maintain 'balance control'. Balance control is necessary because of the sectionalization of the line that leads to different operations being performed at different rates. To provide protection against variations in output over discrete periods of time, an agreed amount of work-in-progress is allowed to act as a buffer between individual operations. A requirement for supervisory control is to set the flow of work through each operation to be as similar as possible. The success of this behavior is reflected in the operational measures of line efficiency and productive performance. Therefore, the primary indicators of unbalanced work flow are idle workstations and declining or overloading work-in-progress buffers.

The monitoring task is necessarily accompanied by analysis. The cause of problems must be identified. Calculations are required to identify which operations require additional resources and which operations can be alleviated of resources.

Formally, a PBS flow line can be viewed as a set of operations, O, where each operation, o, is a 2-tuple which consists of a set of operators, Op, and a set of machines, M ⋆, which have been assigned to the operation:

$$\boldsymbol{o}(\boldsymbol{Op}, M) \quad (1)$$

The scheduling problem consists of assigning operators, op, and machines, m, to an operation, o_1, until the calculated output from the operation is equal, or greater than, the next operation, o_2, in the sequence.

⋆ Op and M have the same cardinality.

$o_1(\{op_1,op_2, .., op_n\}, \{m_1,m_2, ..., m_n\}) \geq 2\ o_2(\{op_1,op_2, .., op_n\}, \{m_1,m_2, ..., m_n\})$.

The sequence constraint, $o_1 \prec 3\ o_2$, is determined by the technical necessity of having to perform operation o_1 before operation o_2.

Other constraints prohibit the operators and machines that can be assigned to an operation. An operation requires a particular skill from an operator, and a particular type of machine. For simplicity we can say that the assigned operators' skill must equal the type of the machine assigned:

$$skill(op) = type(m) \quad 4$$

A typical operator has a set of skills, *S*, operations that she or he can perform, and a particular performance level, *p*.

The process time required for each operation is determined by the work content of the operation, measured by a standard minute value which is empirically determined by a time and motion study, and the performance of the operators. The process time is calculated by dividing the standard minute value of the operation by the performance of the operators and multiplying the result by 100. Operator performance is, again, determined by time and motion study. The industry recognizes that an 100 performer can perform a sewing operation with a 3 minute work value in 3 minutes. Whereas a 50 performer would take 6 minutes.

The maintenance of line balance requires that the process time of each sequential operation is kept as identical as possible. So that $process\_time_1$ and $process\_time_2$ of $op_1(process\_time_1) \prec 5\ op_2(process\_time_2)$ are either equal or within an acceptable range which can be absorbed by the work-in-progress buffers between operations.

Initial Line Balance

The initial line balance is achieved by employing a greedy algorithm which assigns operators to machines and operations.

The greedy algorithm, that assigns the operators and machines to the operations, takes two sets as arguments. The unordered set of operations, *O*. The second argument is an ordered set of skills, *S* ,where skills are constraint satisfaction problem variables, which identify a skill and a domain which consists of the set of operators, *Op*, with that skill. Both the variables, *S*, and the domain values, *Op*, are ordered. The variables are ordered according to the cardinality of the domains. The variables with small domains are considered the most highly constrained and should be processed first. Also, the domains of $s \in 6\ S$ are ordered according to the cardinality of the skill set of the operators. Those operators with less skills are more constrained and should be processed first. The algorithm also takes a variable, *t*, which denotes the target output of the line. The operations, **PUSH**, **POP**, and **REMOVE**, are primitive procedures with obvious interpretations.

The greedy algorithm makes a number of assumptions about hard and soft constraints:

There are sufficient operators to cover the tasks.
There are sufficient machines to cover the task.

- . Operator skills represent hard constraints, and
- . Operator performance represents soft constraints.

Procedure 1

```
PROCEDURE greedy-algorithm (O,S,t)
BEGIN
        WHILE O NOT EMPTY
                DO
                        s  ← 7POP (S)
                        op ← 8POP(Ds)
                        o  ← 9 REMOVE(s,O)
                        PUSH(op,oop)
                        forward_check(op,S)
                      PUSH(machine(s),om)
                UNTIL enough-p (o,t)
      END
END
```

If there are insufficient operators or machines to cover the tasks, it means that the daily output parameter, *t*, must be relaxed. It can be considered to be a soft constraint. The greedy algorithm ignores operator performance, *p*. Operator performance is considered to be a soft constraint which can be relaxed. (The buffers between operations can absorb the additional output of a '100' performer doing a '75' job.) The algorithm only attempts to assign correctly skilled operators to the various tasks.

The added complexity of satisfying performance constraints would require an alternative search procedure, for example, a beam search. This is considered unnecessary because the impact that performance has on output is averaged out over an entire shift. *It is a constraint which can be satisfied over time*. Satisfying operator performance constraints is actually an optimization problem. At any one point in time, the partial satisfaction of the performance constraint is acceptable because the buffers between operations can absorb excess and feed down stream operations. The satisfaction of the operator performance constraint is achieved over

the entire work shift and measured as a constraint which satisfys an objective function.

It is the function of schedule repair to maintain line balance by keeping the process times of operations as equal as possible. This is achieved by transferring, and exchanging, operators between operations. The identification of which operator or operators to transfer or exchange is determined by reassignment heuristics.

Schedule Repair

Schedule repair attempts to solve a dynamic constraint satisfaction problem, and is achieved by reassigning operators to other operations. This activity is supported by partial order backtracking which identifies the appropriate set of candidates.

Schedule repair is triggered by monitoring the flow line. The primary indicators of line unbalance is work-in-progress buffers. An empty (or rapidly emptying) work-in-progress buffer suggests that process times between operations has become unequal. This might be because of human factors. An operator's performance could have dropped because of boredom, or an operator could be absent, or a machine could have broken down. Whatever the reason, the analysis task must generate two sets, the set of those operators, C, whose performance p_{from} can be transferred from it current operation, and the set of operations, T, which must have additional performance, p_{in}. The members of set T are prioritized to identify the most desperate imbalance. Any operator from C whose performance, p_{from}, equals some p_{in} would be an ideal candidate for transfer.

Unfortunately, it is rare that such an ideal candidate can be identified. It is usual that a sequence of exchanges is necessary before a candidate can be freed to add additional performance to an operation.

For example, assume that analysis has identified an operation which requires an additional '75' performance from an overlock operator. The set C of possible candidates does not contain such an operator. However, there is an operator in C with cross stitch skills that can be transferred to a cross stitch operation and allow a '75' performer with both cross stitch and overlock skills to be transferred to the priority operation in T.[†]

The application of reassignment heuristics to rostering problems, and constraint satisfaction problems in general, has been described by Smith (1992)[3].

Partial Order Backtracking: A Discipline for Reactive Rescheduling

The mechanism which supports the selection of exchange candidates is partial order

[†] There is a limit on how recursive these exchanges can be: each time an operator is moved, her or his performance declines and it takes some time before it returns to normal. A flow line which has been seriously perturbed by operator transfers will loose production efficiency. In practice it is better to nominate a small set of operators as floater candidates, and use these exclusively for exchanges.

backtracking. Spragg and Kelleher (1996)[4] have described how 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.

In a recent paper Ginsberg and McAllester (1994)[5] suggested using a hybrid search algorithm that combined the advantages of both systematic and non systematic methods of solving constraint satisfaction problems. The systematic search method described by these authors, dynamic backtracking, employs a polynomial amount of justification information to guide problem solving. The non systematic methods, GSAT (1992)[6] and min conflict (1990)[7],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.

Partial order backtracking brings a systematic search discipline to non systematic schedule repair search procedures, such as GSAT, min conflict and reassignment heuristics, by applying the dynamic backtracking procedure developed by Ginsberg (1993)[8] to the search space.

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

$$10 \quad (x_1 = v_1) \quad \ldots \quad (x_k = v_k) \qquad x \quad v \qquad 11$$

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

$$[(x_1 = v_{1)} \quad \ldots \quad (x_k = v_k) \quad (x = v)] \qquad 12$$

A special nogood is the empty nogood, which is tautologically false. If an empty nogood can be derived from 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:

$$\begin{array}{llll} (x = \boldsymbol{a}) & (y = \boldsymbol{b}) & u & v_1 \\ (x = \boldsymbol{a}) & (z = c) & u & v_2 \\ & (y = \boldsymbol{b}) & u & v_3 \end{array} \qquad 13$$

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:

$$(x = \boldsymbol{a}) \quad (y = \boldsymbol{b}) \quad (z = c) \qquad 14$$

moving z to the conclusion of the above gives:

$$(x = a) \quad (y = b) \qquad z \quad c$$

15

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.

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, a 16, represent a consistent, if partial, solution to a constraint based scheduling problem. 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.

Partial order backtracking 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 procedures takes in exploring the search space.

However, there is not total freedom: safety conditions need to be maintained that model the partial orderings of the variables. It is necessary for variables in the antecedents 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.

Partial order backtracking supplies a framework for reactive rescheduling. The management of a progressive bundle flow line system approaches scheduling as a problem of repair over time. From a constraint based scheduling perspective, rescheduling introduces an extra set of constraints which need to be addressed. These new constraints are related to the need to preserve the old assignments of operators to operations as far 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

An example will demonstrate our approach to reactive rescheduling in a stochastic

environment . We can imagine a progressive bundle flow line with 4 operations:

$o_1 \mathbf{p} o_2 \mathbf{p} o_3 \mathbf{p} o_4$ 17

Where o_1 is an overlock (*ol*) operation with a standard minute value of 2; o_2 is a cross stitch (*xs*) operation with a standard minute value of 2; o_3 is a overlock operation with a standard minute value of 2; and o_4 is a lock stitch (*ls*) operation with a standard minute value of 0.5.

We also have a pool of skilled manual workers (operators) who can be assigned to these operations:

$op_1(S = \{ ol, xs, ls \}, p = 100)$
$op_2(S = \{ ol, xs, ls \}, p = 50)$
$op_3(S = \{ ls \}, p = 75)$
$op_4(S = \{ xs, ls \}, p = 100)$
$op_5(S = \{xs, ls, ol, bt, bh \}, p = 100)$

We assume for simplicity that sewing machines are an unconstrained resource (i.e. when a sewing machine of a particular type is requested, it is always available).

Therefore, formulating this as a constraint satisfaction problem would give the *variables*, o_1, o_2, o_3, and o_4, the following domains (note: ordering heuristics have been applied to both variables and their domains):

$o_4 = \{ op_3, op_4, op_2, op_1, op_5 \}$
$o_1 = \{ op_1, op_2, op_5 \}$
$o_3 = \{ op_1, op_2, op_5 \}$
$o_2 = \{ op_4, op_1, op_2, op_5 \}$

The initial solution to this CSP (using the greedy algorithm described in Procedure 1), $o_{1.1} = op_1$, $o_{1.2} = op_2$, $o_2 = op_4$, $o_3 = op_5$, and $o_4 = op_3$, is supported by the following nogoods:

$o_4 = op_3$
$o_{1.1} = op_1$ $o_{1.2} = op_2$ o_3 op_1 o_3 op_2
$o_3 = op_5$ o_2 op_5
$o_2 = op_4$

18

The operation o_1 is covered by two operators to increase output by reducing the process time.

This initial configuration of the flow line gives the following line performance, measured in the process times (*pt*)‡ of each operation:

---

‡ $\frac{SMV}{p} 100 = pt$

Error! Main Document Only.

$o_1(1.33)$ ᵖ 19 $o_2(2)$ ᵖ 20 $o_3(2)$ ᵖ 21 $o_4(0.6)$

Such a configuration would result in an estimated output of at least 252 garments a day (8 hour work shift)[§]. With this current configuration, it is clear that o_4 can process garments about three times faster than o_3 can supply them, and o_1 can supply garments about one and half times faster than o_2 can process them.

An intuitive solution would be to allow either op_1 or op_2 to 'float' between o_1 and o_3 when necessary to maintain line balance. Such an intuitive, deterministic, solution disregards the problems of environmental uncertainty, machine breakdown, operator absenteeism, declining operator performance because of illness or boredom, etc. etc. The line needs to be monitored, and if there are no perturbations to the work flow we can employ the default strategy of using op_1 or op_2 as 'floaters'.

The identification of either op_1 or op_2 as appropriate candidates for reassignment to o_3 is determined by examining the nogoods for o_1 (i.e. those nogoods which have o_1 as their antecedent):

$$o_{1.1} = op_1 \quad o_{1.2} = op_2 \quad o_3 \quad op_1 \quad o_3 \quad op_2 \qquad 22$$

o_3 appears in the conclusion of the nogood identifying both op_1 and op_2 as suitable candidates for o_3. Which operator is chosen depends upon the analysis of the *performance-analyzer*, a knowledge source which calculates the amount of additional performance required by an operation, or the amount of performance that can be alleviated from a operation.

Let us assume that our analysis suggests that we reschedule the line by reassigning op_1 to o_3, what is the significance of this for the existing schedule? and what are the procedures necessary to discover this significance?

First, we must remove $o_1 = op_1$ from the set of nogoods. We also need to post safety-conditions with the set of nogoods to identify those existing assignments that are now suspect because o_1's assignment of op_1 determined their 'live' domains at the time of their instantiation. These variables are recorded by the nogoods. In our example, o_3 appears in the conclusion of that nogood in which o_1 is the antecedent. The safety-condition, o_1 ᵖ 23 o_3, triggers consistency checks on the reordered variable o_3. We need to determine what affect the reassigning of o_1 will have on its 'future' variables, those variables whose instantiations were determined after o_1 was assigned. Consistency checks show that $o_3 = op_5$ is consistent with op_1 being assigned to o_3. Which is to be expected with a simple 'floater' transfer. Under such circumstances we simply need to reorder and adjust the set of nogoods to reflect the reassignment of op_1:

$$o_4 = op_3$$

$$o_{3.1} = op_5 \quad o_{3.2} = op_1 \quad o_2 \quad op_5 \quad o_1 \quad op_1$$

$$o_1 = op_2$$

$$o_2 = op_4$$

24

[§] This is assuming that o_4 can be maintained with work.

A more interesting situation arises when op_3 is absent. First, we must remove $o_4 = op_3$ from the set of assignments. Here, it is unnecessary to post *safety-conditions*. The ordering heuristics which forced the assignment ensured that no nogoods were posted. An analysis of o_4's domain identifies the ordered set $\{op_3, op_4, op_2, op_1, op_5\}$ as possible candidates for the now unassigned operation. If op_4 is chosen then a substitute for o_2 is also required. Likewise with op_2, a substitute would be required for o_1. Actually, the selection of a candidate operator would be decided, in part, by the *performance-analyzer*. With the current configuration of the line it seems likely this would be either op_1, or op_5. Alternatively, given the performance characteristics, of the operators a sequence of transfers might be recommended by the system implementing the rescheduling framework:

$$op_2 \rightarrow 25\ o_4 \quad \text{and} \quad op_5 \rightarrow 26\ o_1$$

However, before suggesting such a perturbation to the line, with possible consequences on production efficiency, a judgment would need to be made based on the costs and benefits of such an action. Operational matters concerning due dates and job priorities would need to be weighed. This is essentially a managerial judgment. Anyhow, given that the system, and the human user, accepts that op_5 is the ideal candidate, what is the significance of this to the existing schedule, and what procedures are necessary to discover this significance? After removing $o_4 = op_3$ from the set of assignments we reassign op_5 to o_4:

$o_4 = op_5$ o_3 op_5 o_1 op_5 o_2 op_5

$o_3 = op_1$ o_1 op_1 o_2 op_1

$o_1 = op_2$ o_2 op_2

$o_2 = op_4$

The growth in nogood information suggests that perturbations to the line, caused for example by operator absenteeism, is mirrored in perturbations to the ordering of CSP variables and domains. Partial order backtracking allows intelligent reordering of 'past' (previously assigned) variables to allow reassignment heuristics to identify, via nogood sets, suitable candidates for reassignment.

Software Architecture

The software architecture which implements the ideas discussed in this paper is described in Spragg, J., Tyler, D. and Fozzard, G. (forthcoming).[9] The architecture is modeled on the **OPIS** mixed initiative scheduler developed by Smith(1995)[10] and his team at the Center for Integrated Manufacturing Decision Systems, Carnegie Mellon University.

The implementation of the framework described here has been tested in a simulated environment described in Fozzard, Spragg, and Tyler (1996) [11] and Fozzard, Spragg and Tyler (1996)[12].

Conclusion and Future Work

[to be supplied].

Acknowledgments

This paper relates to research undertaken as part of a SERC funded project: The Simulation of Clothing Manufacture. We gratefully acknowledge the contribution of ideas from Matthew Ginsberg, Gerry Kelleher, Steven Smith, and Barbara Smith.

References

[1.] Chuter, A.J., *An Introduction to Clothing Production Management*, BSP Professional Books, Oxford, 1988.
[2.] French, S., *Sequencing and Scheduling, An introduction to the mathematics of the Job-Shop*, Ellis Horwood Limited, 1990.
[3.] Smith, B.M., *Filling the Gaps: Reassignment Heuristics for Constraint Satisfaction Problems*, Report 92.29, School of Computer Studies, University of Leeds, November 1992.
[4.] Spragg, J.E. and Kelleher, G., *A Discipline for Reactive Rescheduling*, Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems, Edited Brian Drabble, The AAAI Press, pages 199-204, 1996.
[5.] Ginsberg, M.L., and McAllester, D.A., *GSAT and Dynamic Backtracking*, Knowledge Representation and Reasoning Conference, 1994.
[6.] Selman, B., Levesque, H., and Mitchell, D., *A New Method for Solving Hard Satisfiability Problems*, In Proceedings of the Tenth National Conference on Artificial Intelligence, pages 440-446, 1992.
[7.] Minton, S., and Johnston, M.D., Philips, A.B., and Laird, P., *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, 1990.
[8.] Ginsberg, M.L. *Dynamic Backtracking,* Journal of Artificial Intelligence Research 1, pages 25-46, 1993.
[9.] Spragg, J., Tyler, D. and Fozzard, G., *FLEAS: flow line environment for automated supervision of simulated clothing manufacture* (forthcoming).
[10.] Smith, S. *Reactive Scheduling Systems*, in *Intelligent Scheduling Systems*, (eds. D.E. Brown and W.T. Scherer), Kluwer Academic Publishers, Boston, Pages 155-192, 1995.
[11.] Fozzard, G., Spragg, J., and Tyler, D. Simulation of Flow Lines in Clothing Manufacture, Part 1: model construction, International Journal of Clothing Science and Technology, Vol 8, No. 4, pages 17-27, 1996.
[12.] Fozzard, G., Spragg, J., and Tyler, D. Simulation of Flow Lines in Clothing Manufacture, Part 2: credibility issues and experimentation, Vol 8, No. 5, pages 42-50, 1996.