17th European Symposium on Computer Aided Process Engineering – ESCAPE17
V. Plesu and P.S. Agachi (Editors)
© 2007 Elsevier B.V. All rights reserved.

Learning to schedule new orders in batch plants using aproximate dynamic programming

Facundo Arredondo, Ernesto Martínez

Instituto INGAR - National Scientific Research Council, Avellaneda 3657, Santa Fe 3000, Argentina, ecmarti@ceride.gov.ar

Abstract

Production scheduling in a wide range of batch plants involves minimizing tardiness of batches already scheduled when inserting new orders. This problem is addressed here as learning an “order insertion policy” using intensive simulations in the framework of approximate dynamic programming (**ADP**). Simple insertion operators are defined and the values of choosing them at different schedule states found by the incoming order are learnt using a *Q*-learning algorithm. To generalize values of insertion operators across schedule states a locally weighting regression technique is used. Results obtained highlight that simulation-based heuristic learning is very appealing to increase responsivenes of scheduling and planning systems in disruptive event handling.

Keywords

Learning, Scheduling, Simulation-based optimization, Supply chain control.

1. Introduction

Although there are many scheduling methodologies grounded on good theoretical foundations [1], the advent of *e*-commerce and the need for automatic handling of disruptive events in the supply chain are emphasizing the development of dynamic (re)scheduling techniques which can truly increase the responsiveness and agility of batch plants [2]. Simulation-based optimization [3] offers a powefull alternative for developing robust policies using reinforcement learning - also known as **ADP**- for ‘on-the-fly’ decision-making in real-time [4,5]. Unfortunately, with few exceptions [3], the use of **ADP** has been almost neglected in dynamic scheduling and planning. It is our contend here that integration of optimization, simulation and learning along with increasing computational power can really make a responsiveness breakthrough in controlling and supervising the supply chain with automatic procedures.

2. Problem Statement

Typically, in most industrial environments when production orders are released to the shop-floor the current schedule should be modified to allow their insertion and a new schedule will result. Batch orders are normally placed on the plant by a central inventory management system; but they also can be placed directly to meet customer needs for special orders. The incoming $(n+1)th$ order has at least three key attributes for its insertion: a product type, a batch size and a due date $\mathcal{D}_{n+1}$. The main criterion for near-optimal insertion is to minimize an arbitrary function of the schedule tardiness:

$$\sum_{i}^{n+1} \tau(\mathcal{D}_i) \tag{1}$$

where $\tau(\mathcal{D}_i)$ is the tardiness of the *ith* order and n is the number of orders in the current schedule (before inserting the arriving order). It´s not included in (1) a penalty term for the disruption caused by rescheduling. Let´s denote by s the vector of variables describing the state of the schedule at the arrival moment of the $n+1$ order and $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_m$ the insertion operators or rules. The entries in s correspond to feature variables that are descriptive of the existing schedule before the arrival of the $(n+1)th$ order. Insertion operators are well-defined procedures to generate a new schedule from the current one following a given rationale and whose degree of optimality, or **value** Q, will depend heavily on the schedule state s. The easiest alternative for defining an insertion operator is doing a 1-batch removal operation according to some criterion, insert somehow the new batch and then reschedule conveniently the earlier removed batch. This removal/insertion/reschedule procedure can be extended to 2-batches, …, n-batches involved in rescheduling. Furthermore, there can be defined more problem-specific insertion operators (see the example below).

Should the value $Q(s_a, \mathcal{O}_m)$ of applying $\mathcal{O}_m$ at the schedule characterized by s_a is known, then the problem of optimal insertion can be easily solved by choosing the operator with highest value at s_a. As this is not the case, Q values must be learnt. In industrial practice, expert shop-floor managers develop over time – through learning - simple rules or insertion heuristics that guide their day-to-day decision making. Using reinforcement learning algorithms [3-5] similar expertise can be learnt using intensive simulations with almost no costs involved, except some computational power. There exist also the need to define meaningful feature variables for schedule states and a reward function to measure the goodness of schedules resulting from applying a given insertion operator. The latter will provide simulated reinforcements in the trial-and-error learning algorithm to drive the initial schedule state to a *goal* state.

3. Simulation-based learning

3.1. Q-learning

One of the most widely used learning algorithm in ADP [3-5] is the one allowing learning directly the values of state-operator pairs. In its simple form, *one-step Q-learning*, is defined by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{2}$$

where r_{t+1} is reward resulting of taking the action a_t at the state s_t and $0<\alpha<1$ is the learning rate. The index t emphasizes the idea of learning in episodes. In our case each learning episode is made up of a sequence of schedule states from the initial state to the *goal* state where transitions are caused by insertion operators. As the values Q of state-action pairs should be incrementally learnt, each operator selection requires a policy with some room for exploration. The easiest alternative is the so-called ε–greedy. With probability (1- ε) the action with highest value is chosen whereas with probability ε any of the non-optimal action is tried. An alternative is the *softmax* criterion based on the current estimates of state-action values. A softmax action selection policy uses a Gibbs, or Boltzmann, distribution with probabilities $\pi(a_t)$ for each action a_t at state s_t:

$$\pi(a_t) = \frac{e^{Q(s,a)/T}}{\sum_{b=1}^{m} e^{Q(s,b)/T}} \tag{3}$$

where T is a positive parameter called the "temperature." High values of T cause actions to be all nearly equiprobable. As learning progresses, T is incrementally lowered which causes a greater difference in selection probabilities for actions that differ in their value estimates.

3.2. State generalization

One problem with the basic *Q-learning* algorithm is the assumption of having a finite number of actions and states. In our application the number of insertion operators is indeed finite but the schedule states are not. To generalize across a continuum of states to *locally weighted regression* (LWR) [7] is used. LWR is a variation of the standard linear regression technique in which training points close to a query point have more infuence over the fitted regression surface. LWR, on the other hand, only fits a function locally, without imposing any requirements on the global form of the *Q*-function. Integration of LWR with the basic *Q-learning* algorithm makes possible to update the *Q*-values of all examples in the region of influence around the query point defined by the kernel parameter κ. This integration dramatically speed up simulation-based learning.

3.3. Order insertion learning

A schema of the algorithm for order insertion learning is given in Fig. 1. Simulations in each learning episode are done using random selection of the initial schedule and new order attributes to account for the widest variety of circumstances in the schedule state found by an incoming order.

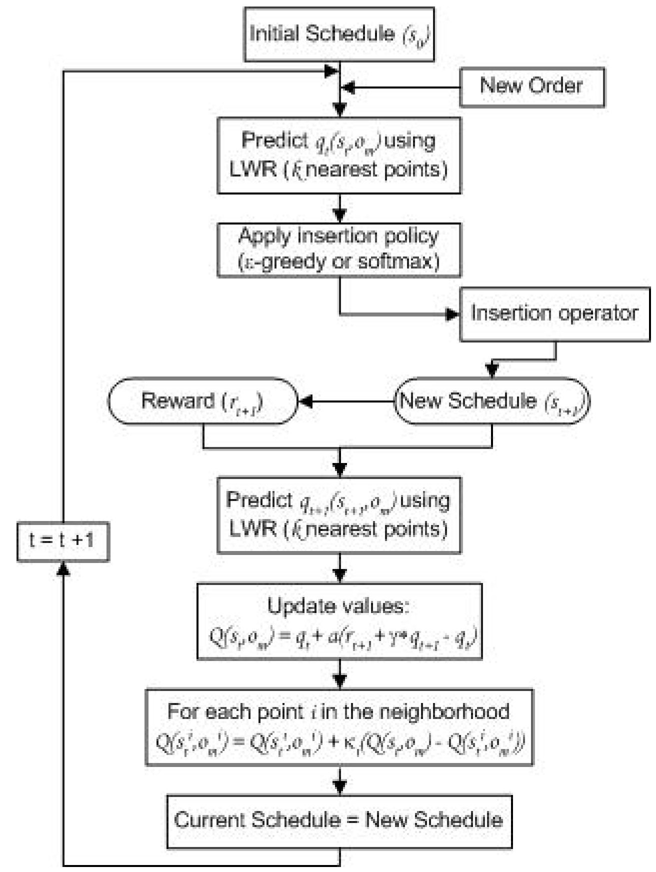
Initial Schedule (s_0)
New Order
Predict $q_t(s_t,o_m)$ using LWR (k nearest points)
Apply insertion policy (ε-greedy or softmax)
Insertion operator
Reward (r_{t+1})
New Schedule (s_{t+1})
Predict $q_{t+1}(s_{t+1},o_m)$ using LWR (k nearest points)
t = t +1
Update values:
$Q(s_t,o_m) = q_t + a(r_{t+1} + \gamma * q_{t+1} - q_t)$
For each point i in the neighborhood
$Q(s_t^i,o_m^i) = Q(s_t^i,o_m^i) + \kappa_i(Q(s_t,o_m) - Q(s_t^i,o_m^i))$
Current Schedule = New Schedule

Fig. 1. Learning to insert orders using operators and *Q-learning*

4. Example

As a rather simple example of the proposed approach, the following single-stage scheduling problem adapted from [6] is used. Plant equipment items are semicontinuous extruders which process orders for 4 products. Processing rates and cleanout requriments are given in Table 1. Also, data for the current schedule and the new order are given in this table. For the sake of space and clarity only three very simple order insertion operators are considered. The simplest one, O_1, seeks the minimum disturbance to the already scheduled order by inserting the new order at the end of one extruder´s queue so that total tardiness is minimized. The operator O_2 allows a bit more room for rescheduling the existing orders by inserting the new order in the position of the extruder´s queue that minimized the total tardiness of the resuting schedule. This operator delays those orders that are left behind by the new order. Finally, the insertion operator O_3 is the 1-batch generic operator designed around the optimal removal/insertion/reschedule for only one batch as discussed earlier. Operator O_3 causes a significant change to the existing schedule, but can provide a nearly optimal insertion. In Fig. 2, the resulting schedules when insertion operators are used to update the existing schedule are shown.

Table 1. A small example problem formulation

Processing rate (kg/day)				
Extruder	Product			
	A	B	C	D
1	100	200	--	--
2	150	--	150	300
3	100	150	100	200
Cleanout requirements (day/cleanout)				
Previous Product	Next Product			
	A	B	C	D
A	0	0	0	2
B	1	0	1	1
C	0	1	0	0
D	0	2	0	0

Customers Orders			
Orders #	Size	Product	Due Date
1	500	A	5
2	100	B	1
3	300	C	2
4	100	C	3
5	700	B	9
6	700	A	7
7	600	D	11
8	300	D	11
9	100	B	6
10	600	B	9
New Order			
11	150	A	5

To learn the values of these three insertion operators using the algorithm in Fig. 1, the following state representation for the schedule found by the (n+1)*th* order is used s=(n, F_1,F_2). The first entry is the number of already scheduled orders, F_1 is the ratio of total late days to total productive days and F_2 is the ratio of total cleaning days to total production days. To assess the goodness or badness r_{t+1} of choosing a given insertion operator the total tardiness (Eq. 1) of the resulting schedule is used, although other criteria would be used as well. Using the estimated Q-values after 10,000 simulations, choosing the insertion operator with maximum value is compared to the actual optimal decision for 2,500 different initial schedules.

The results obtained (see Fig. 3) show that the learning process had converged quickly to a near-optimal insertion policy. Analysis of the operator values reveals that the operator O_1 -which involves no disruption to the existing schedule- is the best decision when there are few cleaning operations in the current schedule. As cleaning time increases, the optimality of O_2 reflects that total tardiness can only be improved by changing the existing schedule. Finally, as cleaning operations are more ubiquitous, insertion is best made by O_3 which involves a complete overhauling of the schedule found by the incoming order.

5. Concluding remarks

The integration of simulation-based optimization with reinforcement learning techniques in scheduling and planning problems opens a new way for automatic control of disruptive events in the supply chain. The problem of near-optimal insertion of a new order in a production schedule was used as a simple example.

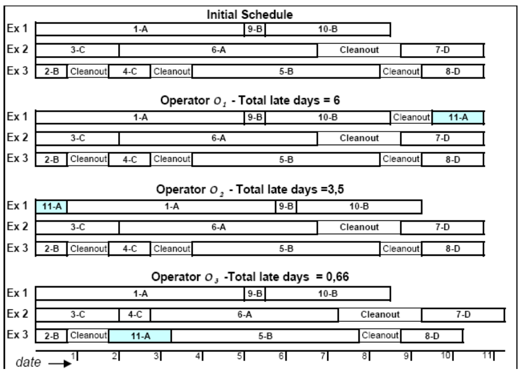
Initial Schedule
Ex 1
Ex 2
Ex 3
1-A
9-B
10-B
3-C
6-A
Cleanout
7-D
2-B
4-C
5-B
8-D
Operator O_1 - Total late days = 6
11-A
Operator O_2 - Total late days =3,5
Operator O_3 -Total late days = 0,66
date
1
2
3
4
5
6
7
8
9
10
11

Fig. 2. Inserting the new order using O_1,O_2 and O_3

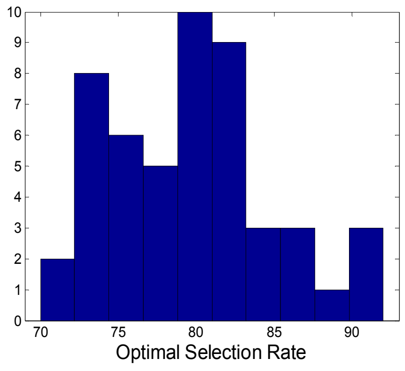

Fig. 3. Using operator values for insertion

References

1. C. A. Méndez *et al.*, *Computers chem. Engng.* **30**, 913-946 (2006).
2. G. E. Vieira, J. W. and E. Lin, *J. of Scheduling* **6**, 39-62 (2003).
3. E. C. Martínez, *Computers chem. Engng.* **23**, S527-S530 (1999).
4. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA (1998).
5. J. Si, A. G. Barto, W. B. Powell and D. Wunsch, *Handbook of Learning and Approximate Dynamic Programming*, Wiley-IEEE Press (2004).
6. R. F. H. Mussier and L. B Evans, *Computers chem. Engng.* **13**, 229-238 (1989).
7. 7.C. G. Atkeson, A. W. Moore and S. Schaal, *Artificial Intelligence Review* **11**, 11-73 (1997).