Production Systems and Information Engineering, Miskolc, Vol. 1. (2003), pp. 83-90

FLOW-SHOP SCHEDULING BASED ON REINFORCEMENT LEARNING ALGORITHM

PÉTER STEFÁN
Computer and Automation Research Institute, Hungarian Academy of Sciences,
Victor Hugo u. 18-22, H-1132 Budapest, Hungary,
Phone: (+36-1) 4503075, Fax: (+36-1) 2709650
stefan@sztaki.hu

[Received September 2002 and accepted May 2003]

Abstract. In the paper a machine learning based method will be proposed to give a quasi-optimal solution to the *m*-machine flow-shop scheduling problem. Namely, given a set of parts to be processed and a set of machines to carry out the process and the sequence of machines is fixed, each part should have the same technological path on all machines; the order of jobs can be arbitrary. The goal is to find appropriate sequence of jobs that minimizes the sum of machining idle times.

1. Introduction

Recent research has put increasing emphasis on scheduling in all production phases. The goal of scheduling is defined as finding the best sequence of different activities (processing operations, delivering the goods) given a set of constraints imposed by the real world processes. These constraints can cover physical laws as well al rising costs that can make production unrealistic or uneconomic.

Basically, there are three main scheduling concepts: mathematically grounded algorithms, heuristic approaches and algorithms supported by machine learning (ML). The first concept can be adapted to small-sized scheduling problems. Johnson's algorithm, e.g., solves a two-machine flow-shop scheduling task that is established by classical algebraic and dynamic programming ways as well. The advantage of the algorithm is that it is well defined, exact and can be generally applied to the wide range of two-machine scheduling tasks. The price is lack of scalability: i.e. no mathematical proof can be given for a larger number of machines.

As a potential improvement, there are two directions of research to overcome the restrictions of mathematical formulations: using heuristics, and/or machine-learning. Both directions try to set up some model of human being problem-processing capability but in different ways. While heuristic approaches provide direct rules of thumb to follow, but no algorithm to find the solution in a modified decision environment. ML methods give a model of a mental process itself. As the knowledge of the learning agent improves the method results in solutions that are more and more close to the optimal solution, even in changing environment.

In the paper an algorithm will be shown that is capable of "learning from scratch" using a reward-punishment procedure, called reinforcement learning (RL) [4].

2. Scheduling task

The scheduling problem, under consideration, is called flow-shop scheduling where given a set of parts to be processed (jobs) and a set of machines for processing. Each part has the same technological path on all machines; the order of jobs is arbitrary. The goal is to find the appropriate sequence of jobs that minimizes the sum of idle times.

2.1. Johnson's algorithm

Let jobs be denoted by $j_1, j_2, \ldots, j_n$ while the two machines by A and B. If there are no precedence restrictions among the jobs, there is $n!$ (n factorial) number of possible job-sequencing on the two equipment, which yields non-polynomial (NP) hard task. Figure 1 illustrates the Gantt diagram of one possible job sequence.

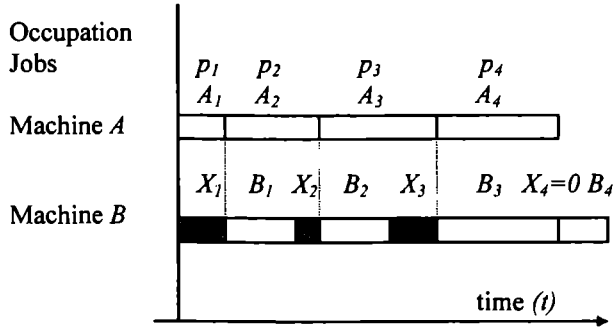

Figure 1 Job sequence on two-machines. p_i denote parts, A_i machining times on A, B_i machining times on B and X_i are idle times on B.

In the figure, $X = \sum_{j=1}^{n} X_i$ indicates the total off-machining time on machine B, as well as the total idle time of the scheduling task with two-machines. The goal of optimization is to find a job-order, which minimizes X.

The possible data structure of the algorithm, for example, is an $n \times 2$ matrix which can be seen in Figure 2. The scheduling method itself is illustrated in Figure 3. The proof of the algorithm can be found, e.g., in [1].

2.2. Three-machine extensions: Palmer's and Dannenbring's methods

The two-machine scheduling algorithm can be extended to three-machine scheduling by imposing additional restrictions on machining times.

A method that was first published by Palmer uses job priorities to set up job sequences. Single priority value compresses the following concept in a single rule: jobs that produce

shorter execution on the first machine are sorted to the beginning; jobs that have shorter execution times on the third machine are left behind [5].
Dannenbing's algorithm decomposes the *m*-machine scheduling task to *m*-1 two-machine tasks compromising quasi-optimal values [6].

	A (minutes)	B (minutes)
Job1	10	15
Job2	20	15
Job3	30	65
Job4	20	10
Job5	45	100

Figure 2 Machining times of different jobs on machines *A* and *B*. The length of machining is measured e.g. in minutes

```
function Johnson (table of machining times)
                                        return optimal sequence
  let Q denote the queue of jobs
  initialize Q with empty set
  for each j cell of the table {
    find the minimal machining time scanning in both columns
    if the time found occurs in column A then
      add the job_j to the beginning of the queue Q
    else add job_j to the end of the queue Q
      delete the slot found
    endif
  return Q as the optimal sequence
end
```

Figure 3 Johnson's algorithm

3. Machine learning approach

In order to implement any machine-learning algorithm, first, the concept of learning should be defined. Learning, in its original sense, means that a system is capable of modifying its internals (structure or parameters) to satisfy the requirements of the "evaluator" (or teacher, or external environment).

3.1. Determination of the optimality criterion

For translating the definition into scheduling terms, some evaluation process has to be developed to be capable of distinguishing among different schedule plans (evaluator, or fitness function in genetic algorithms).

As the definition of scheduling environment is clear, it is easy to develop an evaluation algorithm to express the "goodness" of scheduling in numerical terms. Figure 4 shows an algorithm that computes idle times to an arbitrary number of machines and jobs. The real benefit of the method is that it computes partial idle time data corresponding to individual machines and jobs.

The evaluator inputs the matrix of machining times (**M**) and the job and/or machining sequence permutation vectors (**r**, **p**). Both vectors are necessary for the proposed scheduling algorithm, not by the scheduling task itself. The number of jobs is indicated by n, the number of machines by m. The algorithm outputs a scalar value v indicating the sum of idle times, vector **d** stores sum of idle times preceded by the corresponding job.

The computation cost of evaluation is at level $O(nm)$. More details about the algorithm can be found in [5].

```
input M[m,n], r[m], p[n];
output v;
storage d[n], D[m,n];
function n_machine return D[m,n] or v
begin
  v:=0;
  for i:=1 to m do
    s[i]:=M[r[i],1];
    d[i]:=v;
    D[i,1]:=v;
    v:=v+M[r[i],1];
  end
  for j:=1 to n do D[1,j]:=0;
  for j:=2 to n do
    for i:=1 to m-1 do
      s[i]:=s[i]+M[r[i],p[j]];
      D[i+1,j]:=max(0,s[i]+d[i]-s[i+1]-d[i+1]);
      d[i+1]:=d[i+1]+D[i+1,j];
    end
    s[m]:=s[m]+M[r[i],p[j]];
  end
  v:=0;
  for i:=1 to m do v:=v+d[i];
  return v;
end
```

Figure 4 Determination of machining idle times for arbitrary number of machines and jobs

3.2. The learning module

Having the evaluation algorithm been examined, a question can be immediately addressed: how can it be used in real-life machine learning applications? There are two basic approaches under consideration: reinforcement learning and genetic algorithm.

One of the possible approaches is to use a reinforcement-learning (RL) based module, called Q-learning to maintain job precedence preferences, or in RL terms, action-state values.

```
input M[m,n], jobs, alpha, gamma;
output r[n];
storage Q[n][n], V[n];
function Q-update return r[n];
begin
  Q[i][j]=0 for all i=1..n, j=1..n;
  while annealing_cycle do
    r[n]=permutation(jobs);
    reward=n_machine(M[m,n],r[n]);
    for i:=1 to n-1 do
      update_Q_table(Q[r[i],r[i+1]], reward,alpha,gamma);
    end
    update_V(r[1],reward,alpha,gamma);
  end
end
```

Figure 5 Q-learning based flow-shop sequencer algorithm

RL methods evolved from dynamic programming and automata theory and model reality through a set of states, state-changes and values (preferences) assigned to both. RL also defines update rules over the state, state-change model, which are used for maintaining values with respect to the measured feedback provided by the environment of the learning model.

Whenever an RL method is applied to a certain problem, the property that can be used as states should be identified. Furthermore, the state-changes and the reward measurement should also be identified. Then the RL algorithm makes explorative and exploitative traverses in the state-space trying to find a path that is highly rewarded. The benefit of the algorithm is its capability of exploration, i.e. traversing through states that are not well-rewarded but may yield higher reward in the long run, bypassing local maxima this way. It is important to pay attention to exploitation and exploitation balancing problem [4]. Exploration is interpreted as an operation mode of the learning agent when it makes experiments and tries to discover its environment. On the other hand, exploitation is a mode when the agent has gathered enough knowledge and makes real decisions.

In the flow-shop scheduling exercise the model takes machining times, machining costs as input parameters, and job sequence as a variable parameter, and a certain job sequence is sought that minimizes idle time, in the long run.

To fit RL methods, it is reasonable to define states as job sequences, or more precisely job precedence relations. State-changes (or actions) are defined as changes in relations. An action step is performed by a permutation operator, which sets up a job sequence according to precedence preferences. At the beginning no preferences are given, so states are traversed randomly. As learning proceeds, preferences are updated, which, in turn, influences action selection policy converging to the found quasi-optimal job sequence. From this respect the learning algorithm is a directed search procedure.

Parameters of the algorithm are the same as those of the evaluation algorithm, except RL specific arguments: **Q** stores action-state values (decision preferences), alpha and gamma regarding to learning rate and discount rate [2].

The Q-learning based algorithm can be seen in Figure 5. It introduces a new element, **v**, which is a vector of preferences. Array **v** expresses the value of starting the job sequence with job_i, for all jobs. Update methods are formulated as the usual RL update rules:

$$Q_{n+1}(j_i, j_j) = Q_n(j_i, j_j) + \alpha(r + \gamma \max_k Q_n(j_i, j_k) - Q_n(j_i, j_j)) \quad (1)$$

$$V_{n+1}(j_i) = V_n(j_i) + \alpha(r + \gamma \max_k Q_n(j_i, j_k) - V_n(j_i)) \quad (2)$$

Evaluation algorithm shown in Figure 4 can be used as a fitness evaluator of any genetic algorithm-based method as well. In this case job sequences are regarded as "fenotypes", heuristic operators such as mutation and crossover are defined as specific job permutations.

4. Implementation

An RL-based scheduler has been developed in Java in order to validate theoretical results. The code is formulated to be modular to let non-RL modules be plugged in and allow comparisons between them.

Figure 6 shows the screenshot the Gantt chart of an example five-machine, nine-job scheduling task.

So far the scheduler is capable of learning using step-by-step iteration, and through a single annealing period. Annealing schedule[1] is set up manually. Algorithm in Figure 4 has been extensively tested on different job-machine setups. The accuracy of the learning procedure depends on the "speed" of the annealing schedule: the larger the annealing period is, the more accurate the solutions are. Given a time horizon three annealing schedules have been compared. The best result was achieved when concave annealing function was chosen, which let exploration until about 95% of the full time horizon and "chopped down" exploration turning immediately into exploitation phase.

The optimality criterion can be modified easily: multiple optimum criteria can also be applied provided that it can be mapped into a single scalar feedback.

5. Conclusions and future work

It has been established that RL-scheduler is able to find close-to-optimal solution, and RL combined with simulated annealing and balancing algorithms are also capable of finding quasi-optimal solutions when machining-times vary in time.

[1] The term annealing schedule regards the temperature-time function and has nothing in common with machining schedule.

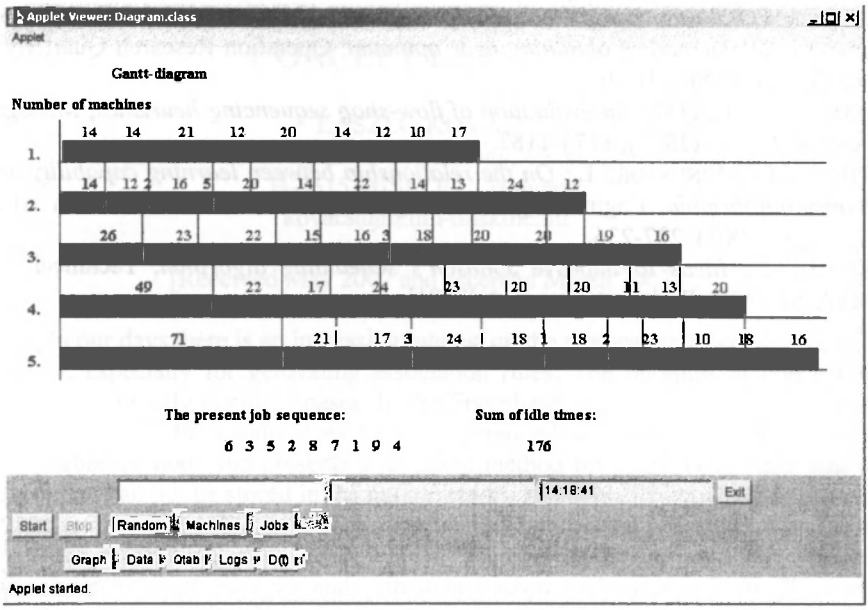

Figure 6 Gantt-chart of a 9-job, 5-machine task in a RL-based simulator, numbers above stripes indicate processing or idle times

As for the future plans, it is purposed to make a detailed comparison among the results of a RL-scheduler; a genetic algorithm-based scheduler and the Johnson algorithm. A new protocol is also under development which provides a real manufacturing environment-virtual manufacturing environment communication primitive, which can be used for on-line learning.

Acknowledgements: The author would like to express his gratitude to Prof. László Monostori, Prof. László Dudás, Prof. Ferenc Erdélyi and Prof. Tibor Tóth for their help and inspiration.

References

[1] TÓTH, T.: *Design and Planning Priciples, Models and Methods in Computer Integrated Manufacturing*, Publisher of the University of Miskolc, (1999), 252 p. (in Hungarian).

[2] MONOSTORI, L., MÁRKUS, A. VAN BUSSEL, H, WESTKAMPFER, E.: *Machine learning approaches to manufacturing*, Annals of the CIRP, (1996), pp.675-712.

[3] TETI, R., KUMARA, S.R.T.: *Intelligent computing methods for manufacturing systems*, Annals of the CIRP, (1997), pp.1-24.

[4] SUTTON, R., BARTO, A.: *Reinforcement Learning (An Introduction)*, The MIT Press, Cambridge, Massachusetts, (1998), 312 p.

[5] PALMER, D.S.: *Sequencing jobs through a multi-stage process in the minimum total time-a quick method of obtaining near optimum*, Operation Research Quarterly, Vol. 16, No. 3, (1965), 101-107.

[6] DANNENBRING, D.G.: *An evaluation of flow-shop sequencing heuristics*, Management Science 23(11), (1977), 1174-1182.

[7] STEFÁN, P., MONOSTORI, L.: *On the relationship between learning capability and the Boltzmann-formula*, Engineering of Intelligent Systems, Lecture Notes in AI 2070, Springer, (2001), 227-236.

[8] STEFÁN, P.: *Ideas to improve Johnson's scheduling algorithm*, Technical Report, MTA-SZTAKI, Budapest, Hungary, (2001).