ASYMPTOTIC SCHEDULING

by
Paolo Serafini, University of Udine, Italy
Dipartimento di Matematica e Informatica, Via delle Scienze 206, 33100 Udine, Italy

Abstract: We deal with the following scheduling problem: a finite set of jobs is given and each job consists in the execution of an infinite number of tasks. A task is a sequence of operations and each operation requires a specific machine. A machine can process only one operation at a time and preemption is not allowed.

Performance measures of the processing system involve fixing a time horizon T, counting the number of tasks completed within T for each job and maximizing a specified function of these numbers to estimate the throughput of the schedule. Whilst computing the throughput for a given T is in general an extremely difficult problem, it is shown in this paper that the limit, as T tends to infinity, of the average throughput (i.e. the throughput divided by T) can be easily computed via Linear Programming under fairly mild conditions. This quantity, which may be called the asymptotic throughput, can be used to assess a bound on performance measures of real systems. Buffers play a crucial role and buffer sizes can be taken care of in assessing the system performance.

Keywords: scheduling, linear programming, production throughput.
MSC: 90B35, 90C05, 90C27, 90C90.

1. INTRODUCTION

In this paper we deal with scheduling problems in which very large volumes of identical items have to be processed and there is a continuous flow of operations from the input to the output of the processing system. This is typical of many real production environments where one is interested in producing the largest number of items per time unit through a balanced exploitation of all available resources within the typical scheduling constraints.

In the scheduling problems we consider, processing an item requires a particular resource which we simply call machine. We also take into account the presence of physical locations, so called buffers, where the items wait before processing.

In this framework it is important to be able to assess the best productivity of the system, to identify possible bottlenecks and/or idle parts, and consequently to redesign the system in order to gain a better performance. This is clearly a formidable task if the time horizon is finite. Surprisingly, with an infinite time horizon this computation becomes easy for a wide class of problems.

The main result of this paper is that computing the so called asymptotic throughput of the system (the "best" asymptotic ratio of produced items per time unit) can be simply done via Linear Programming. The Linear Programming solution consists also of the average frequencies of the operations and a schedule achieving the asymptotic throughput can be realized by a simple assignment rule based on these values.

The rationale behind this result is that as the time horizon goes to infinity the machines are fully decoupled by the buffers, so that the strong interaction between different resources, which makes typically difficult a scheduling problem, disappears. Hence a crucial feature is the size of the buffers. If the buffers are sufficiently large the asymptotic schedule can be implemented. However, if the buffers do have bounds, a linear constraint can be added to the model thereby providing a more realistic asymptotic throughput.

At the best of the author's knowledge no similar result has appeared in the literature. No mention can be found in state-of-the-art reviews like Ball et al. (1995 a, 1995 b), Graves et al. (1993), texbooks like Silver et al. (1998) or review papers like Pochet (2001), although at the end of this paper there are some connections with the subject of our paper. Maybe models of this type, simply involving a linear programming model, have been already used but not published. However, we point out that the results of this paper should not be regarded as a practical tool to build a detailed schedule. Their value is more theoretical in the sense that they can provide a simple way to compute an ideal (i.e. asymptotical) system performance, against which the real performance can be evaluated possibly receiving suggestions to modify the system design. The ideas of this paper originate from the result in Papadimitriou et al. (1993) where a simpler communication problem is investigated.

2. MATHEMATICAL FORMULATION

We deal with the following problem: a finite set J of jobs and a finite set M of machines are given. A job consists in the execution of an infinite number of tasks, which are all associated to that job. Each task consists of a finite sequence of operations. The sequence is not necessarily the same for all tasks of a job and, in case there are different sequences, a task can be assigned to any one of them, since these alternative sequences lead to the same final 'product'. Each operation a is executed by a specific machine m with a definite processing time p_a. A machine can process only one operation at a time and preemption is not allowed. If necessary, we may introduce dummy operations in the model with arbitrary processing times.

The set of the various sequences of operations for the tasks of a given job is finite and is called a routing. In the simplest case a routing consists of a single operation (the same for all tasks), which is represented as an arc directed from a source node s to a sink node d. In a more complex case it may consist of several alternative sequences of operations, which are represented as several disjoint paths from a single source node s to a single sink node d, with arcs corresponding to operations (all operations on the paths are viewed as distinct in this model even if some of them correspond to the same real operation).

Routings can be also defined implicitly by taking the set of all paths from a source node s to a sink node d on a directed graph. By this we mean that the arcs of any path from the source to the sink correspond to a feasible sequence of operations. The previous case of a simple arc can be viewed as the extreme case of a directed acyclic graph consisting of one single arc.

Hence we associate to each job $j \in J$ the graph G_j, representing the routing of job j, obtained by identifying all previously defined sources and sinks as a single source node s_j and a single sink node d_j. Let $G = \cup_{j \in J} G_j$. Note that G it is not connected if there is more than one job. Its connected components are the routings G_j. We recall that a task can be executed by any directed path of operations in G_j.

Let V be the set of nodes of G and A the set of arcs, or, equivalently, operations. For each $v \in V$ let us denote by v^- and v^+ respectively the sets of arcs incident to and from the node v. Let V' be the set of nodes obtained by excluding the sources and the sinks. The set A can be partitioned in two alternative ways. The first one refers to the jobs, so that we may write $a \in j$, with $j \in J$. The second one refers to the machines which execute the operations, so that we may write $a \in m$, with $m \in M$.

The execution of a task may typically require 'moving' an item between machines according to the particular routing. For instance such an item can be a physical piece to be worked by some manufacturing

process or it can be a sequence of bits representing information to be processed. Such an item is associated with some task and there may be idle times between consecutive operations of that task. During the idle times the item must be located in some places, called buffers. In this model we identify a buffer with a subset $B \subset V$, with the idea in mind that B refers to a single definite physical location. All items associated with tasks routed through nodes $v \in B$ are located in the same buffer. A node can belong to at most one buffer and there may be nodes not associated to any buffer.

For a fixed time horizon T, a schedule specifies for each job j the paths in G_j of its tasks completed within T and the starting times of the operations for these tasks. A schedule is feasible if, for each task, the starting times of its operations are consistent with the path sequence of the task and the processing times, and, for each machine, the operations associated to the machine do not overlap in time.

Given a feasible schedule we may compute the step functions $n_j(t)$, defined as the number of tasks of job j completed within time t, and $n_a(t)$, defined as the number of times the operation a has been completed within t. The functions $n_a(t)$ increase by one at the completion times, i.e. if t is the completion time of an operation a we have $n_a(t-\varepsilon) = n_a(t) - 1$ for any $\varepsilon > 0$ sufficiently small. Feasibility of a schedule implies feasibility of the following constraints for all $t \leq T$:

$$\begin{aligned}
&\sum_{a \in m} n_a(t)\, p_a \leq t && m \in M \quad \text{(capacity constraints)} \\
&\sum_{a \in d_j^-} n_a(t) \quad = n_j(t) && j \in J \quad \text{(job constraints)} \\
&\sum_{a \in v^-} n_a(t) \quad \geq \sum_{a \in v^+} n_a(t) && v \in V' \quad \text{(balance constraints)}
\end{aligned} \tag{1}$$

Let us comment on the constraints expressed in (1). The capacity constraints follow from the fact that the total execution time of the operations allocated within time t on the same machine must not exceed t. The job constraints relate the tasks of each job to the different routings. Finally for each routing the number of operations having a certain node as starting point cannot exceed the number of operations having the same node as end point, since each routing can be viewed as a flow through the network. This is expressed in the balance constraints.

Note that feasibility of (1) is only a necessary condition for a feasible schedule. The mere existence of non decreasing step functions $n_a(t)$ and $n_j(t)$ feasible in (1) for all $t \leq T$ does not imply that the schedule implicitly derived from $n_a(t)$ and $n_j(t)$ is feasible. Indeed we may build functions such that $n_a(T)$ and $n_j(T)$ are feasible (and as large as possible) with $n_a(t) = 0$ and $n_j(t) = 0$ for $0 \leq t \leq T-1$. This is as having all production concentrated in one time unit which is clearly impossible in general.

Furthermore, feasibility of just the values $n_a(T)$ and $n_j(T)$ does not imply existence of a feasible schedule yielding these values. Consider two jobs, the first job consisting of the operation a_1 followed by a_2 and the second job of the operation a_3 followed by a_4. The operations a_1 and a_4 are associated to machine m_1 with processing times $p_1 = 2$ and $p_4 = 4$ and the operations a_2 and a_3 are associated to machine m_2 with processing times $p_2 = 3$ and $p_3 = 1$. Then by taking $T = 10$ the values $n_1(T) = n_2(T) = 3$ and $n_3(T) = n_4(T) = 1$ do respect the capacity constraints $2\,n_1(T) + 4\,n_4(T) \leq T$ and $3\,n_2(T) + n_3(T) \leq T$ but it is impossible to complete within $T = 10$ three tasks of the first job and one task of the second job.

The striking fact we shall later prove is that feasibility of a set of conditions of the type (1) is sufficient to guarantee the existence of a feasible schedule if the time horizon is infinite.

We measure the quality of a schedule through the numbers $n_j(T)$ (let $n(t) := \{n_j(t)\}_{j \in J}$ be the array of all $n_j(t)$). Roughly speaking the larger the numbers $n_j(T)$ are the better the schedule is. For our purposes it is not necessary to fully specify a performance measure. We may consider a generic objective

function $f(n(T))$ which we only assume to be continuous, non decreasing and positively homogeneous, i.e. $f(c \cdot n(T)) = c \cdot f(n(T))$ for $c > 0$.

Typical examples of objective functions of this type are: $\min_{j \in J} n_j(T)$ (balancing the number of tasks of each job), $\sum_{j \in J} n_j(T)$ (striving for the maximum total number of tasks irrespective of the jobs), $\min_{j \in J} n_j(T)/w_j$ (balancing a mix of tasks proportional to the values w_j), $\sum_{j \in J} n_j(T)/w_j$ (striving to the maximum total number of "weigthed" tasks). It may be surprising that we require $f(\cdot)$ to be continuous when its arguments are in fact integer numbers. Later we shall use the same function f with real arguments converging to a limit and continuity will be essential to prove our thesis.

We call *throughput* the maximum of $f(n(T))$ with respect to all feasible schedules. We denote the throughput by $F(T)$. We also define the *asymptotic throughput* as $F := \lim_{T \to \infty} F(T)/T$. We are mainly concerned with the problem of finding the asymptotic throughput and we also briefly address the problem of finding the throughput for a finite horizon T.

It may be helpful to gain a better understanding of the previous concepts by showing two examples (one more example can be found in Papadamitriou et al. (1993) where a communication problem is investigated).

Example 1: This is taken from a real flexible manufacturing system. A set of presses is available for the job of shaping identical containers. The pressing process requires a certain time p_h depending on the press h. Each produced item is carried away by one AGV (automatic guidance vehicle) with transportation time depending only on the location of the press with respect to a central depot. Let q_h be the time to carry out a mission from the depot to the press h and back. Then a task is a sequence of two operations: a pressing operation and a transportation. The number of alternative sequences is clearly given by the product of the number of presses times the number of AGV's.

To model this problem in the stated framework we have to define the graph G (we are supposing a single job and therefore G is just the routing of the job). One possible way to do it is to build a complete directed bipartite graph $(H, K; E)$, with node $h \in H$ associated to the press h and node $k \in K$ associated to the AGV k, to add a source s linked to all nodes in H and a sink node d linked to all nodes in K (arcs directed to d). The arcs (s, h), $h \in H$, represent the pressing operations and we associate the processing times p_h to them. The nodes in H represent the press buffers after pressing. The arcs $(h, k) \in E$, $h \in H$, $k \in K$, represent the transportation operation of AGV k carrying away an item from press h and we associate the transportation time q_h to the arc (h, k). The arcs (k, d), $k \in K$, correspond to dummy operations and the nodes in K are not associated to any buffer (this is because of the dummy operations).

Let $n_h(t)$ be the number of items produced by press h within time t (so that $n_h(t)$ counts the operations on the arcs (s, h), $h \in H$), $n_{hk}(t)$ be the number of completed missions of AVG k to press h (i.e. $n_{hk}(t)$ counts the operations on the arcs $(h, k) \in E$), $n_k(t)$ be the number of completed missions of AGV k (i.e. $n_k(t)$ counts the operations on the arcs (k, d), $k \in K$) and $n(t)$ be the total number of items carried to the central depot. So the constraints (1) become:

$$
\begin{aligned}
p_h\, n_h(t) &\le t && h \in H && \text{(capacity constraints for the presses)} \\
\sum_{h \in H} q_h\, n_{hq}(t) &\le t && k \in K && \text{(capacity constraints for the AGV's)} \\
\sum_{k \in K} n_k(t) &= n(t) && && \text{(job constraint)} \\
n_h(t) &\ge \sum_{k \in K} n_{hk}(t) && h \in H && \text{(balance constraints on } H\text{)} \\
\sum_{h \in H} n_{hk}(t) &\ge n_k(t) && k \in K && \text{(balance constraints on } K\text{)}
\end{aligned}
\tag{2}
$$

and we are clearly interested in maximizing $n(t)$. ∎

Example 2: This is another real case for which the model has been applied. In a small clothing factory fabric need to be cut, sewn, hemmed, ironed etc. Each article (i.e. the final cloth) may be processed in a few alternative sequences of the stated operations. Usually an operation may be carried out by any machine within a specific subset of machines with different performances. Suppose an article requires first an operation on a machine of type A, then two operations in any order on machines of type B and C respectively and finally an operation on a machine of type D. This gives rise to the following routing: an arc for the operation, labeled 1, on machine A, followed by two parallel paths of two arcs each (operations 2 and 3 on the machines B and C respectively on one path and operations 4 and 5 on the machines C and B on the other path); the two paths merge into the arc for the operation 6 on machine D. Then each arc is turned into a set of parallel arcs corresponding to the alternative identical machines of the same type available for the operation. Supposing that this is the only routing in the production process and that all machine types are available in two identical units, we have (operation i on the h-th machine unit is labeled ih; $n(t)$ is the number of produced articles):

$$\begin{aligned}
n_{1h}(t)\, p_1 &\leq t && h := 1,2 \\
n_{2h}(t)\, p_2 + n_{5h}(t)\, p_5 &\leq t && h := 1,2 \\
n_{3h}(t)\, p_3 + n_{4h}(t)\, p_4 &\leq t && h := 1,2 \\
n_{6h}(t)\, p_6 &\leq t && h := 1,2 \\
n_{61} + n_{62} &= n(t) \\
n_{11} + n_{12} &\geq n_{21} + n_{22} + n_{41} + n_{42} \\
n_{21} + n_{22} &\geq n_{31} + n_{32} \\
n_{41} + n_{42} &\geq n_{51} + n_{52} \\
n_{31} + n_{32} + n_{51} + n_{52} &\geq n_{61} + n_{62}
\end{aligned}$$

The first eight inequalities are the capacity constraints for machines of type A, B, C and D respectively, the equality is the job constraint and the last four inequalities are the balance constraints in the four nodes of the routing.

In this example buffers are not directly related to the nodes since the arcs outgoing from a node refer to operations on different machine units (and also in one case to different machine types). If there is the need to consider buffers feeding machine units, we have to introduce dummy operations whose purpose is to 'split' each node into several nodes, one for each machine unit: each arc becomes two arcs in series, the first arc being the dummy arc (an 'assignment' operation), the second being the actual operation. ∎

3. FINITE THROUGHPUT

The problem of finding an optimal schedule with a fixed time horizon T is a very difficult one. Let us consider the recognition version of the problem:

Finite Throughput Problem: Given a time T and a constant K is there a feasible schedule such that, for all tasks completed within time T, the value of the objective function is not less than K ? ∎

A wide class of scheduling problems are particular cases of the Finite Throughput Problem. For instance if one takes as objective function $f(n(T)) = \min_{j \in J} n_j(T)$ and sets $K = 1$, then the question is concerned with the existence of a schedule such that at least one task for each job is completed within time T. This is indeed the usual setting of many scheduling problems for which the focus is on the possibility of carrying out only one run of tasks, within certain time prescriptions.

Among these problems let us consider the Job Shop problem: jobs are given, each job consisting of a chain of operations, and each operation is executed by a definite machine with a known processing time (for a more detailed description of the problem see for instance Applegate and Cook (1991)). In the optimization version the goal is to minimize the makespan and in the recognition version the goal is to complete one run of jobs within a stated time T. Clearly this problem is a particular case of the Finite Throughput Problem with $f(n(T)) = \min_{j\in J} n_j(T)$, $K = 1$ and each job chain of operations in the Job Shop problem being obviously a path G_j. Since the Job Shop problem is NP-hard (Garey and Johnson (1979)) this suffices to show that the Finite Throughput Problem is NP-hard. Just note that the graph G of our model differs from the usual disjunctive graph of the Job Shop problem where operations are associated to the nodes whilst in our model activities are associated to the arcs. Moreover, there are no disjunctive arcs in our model and the capacity constraints are taken care of in the mathematical programming formulation and not in graphical terms.

The question is whether the Finite Throughput Problem is in NP. The fact that we are dealing with more than one run of tasks introduces a new degree of difficulty. Indeed we may think of a certificate for an yes-instance of the problem which merely lists the starting times for all operations. The length of this list is certainly linearly bounded by K since the objective function is positively homogeneous, but this bound is not polynomial in the size of K. We conjecture that the Finite Throughput Problem is P-SPACE complete. It is not difficult to show that it is in P-SPACE (we may follow the same line of reasoning as in Papadamitriou et al. (1993)).

4. ASYMPTOTIC THROUGHPUT

It has been previously remarked that a feasible schedule is characterized by satisfaction of the constraints (1) for all values of $t \leq T$ and that a feasible solution of (1) for the unique value $t = T$ does not in general corresponds to a feasible schedule. Now we are going to prove that a simple set of linear constraints derived from (1) is sufficient to solve the problem of finding a schedule which maximizes the asymptotic throughput.

Indeed there is the following intriguing fact: whereas the problem for the finite horizon case is very difficult, as the time horizon goes to infinity the problem can be simply solved by computing through a mathematical programming problem the average frequencies of the operations and by implementing the schedule through a simple assignment rule.

As the time horizon T tends to infinity we define the following variables $x_j := \lim_{T\to\infty} n_j(T)/T$ and $x_a := \lim_{T\to\infty} n_a(T)/T$, which correspond to the average frequency for the execution of a task of the job j and of the operation a respectively. Let $x := \{x_j\}_{j\in J}$. Now we derive from (1) the following mathematical programming problem, which can be easily modeled as a linear problem for the type of functions f previously shown (i.e. $\min_{j\in J} x_j$, $\sum_{j\in J} x_j$, $\min_{j\in J} x_j/w_j$, $\sum_{j\in J} x_j/w_j$), which account for the vast majority of interesting cases:

$$
\begin{array}{lllll}
\max & f(x) & & & \\
& \displaystyle\sum_{a\in m} p_a\, x_a \;\leq\; 1 & & m \in M & \text{(capacity constraints)} \\
& \displaystyle\sum_{a\in d_j^-} x_a \;=\; x_j & & j \in J & \text{(job constraints)} \\
& \displaystyle\sum_{a\in v^-} x_a \;=\; \sum_{a\in v^+} x_a & & v \in V' & \text{(balance constraints)} \\
& x_a \;\geq\; 0 & & a \in A &
\end{array}
\tag{3}
$$

Let $\hat{F}$ be the optimal value of (3). We want to show that $\hat{F} = F$. We recall that F has been defined as the asymptotic throughput.

Lemma 1: $\hat{F} \geq F$.

Proof: Given any schedule and for any time T we may find a feasible solution for (3) in the following way: let $n_j(T)$ be the number of completed tasks at time T, and let $n_a(T)$ be the number of completed operations, relative to the completed tasks only. Let us define $x_j := n_j(T)/T$ and $x_a := n_a(T)/T$. Since clearly $\sum_{a \in m} n_a(T) p_a \leq T$, x_a satisfies the capacity constraints in (3). By definition also the numbers $n_j(T)$ and $n_a(T)$ must be such that the job and balance constraints are satisfied. The value $f(x)$ of this feasible solution is such that $\hat{F} \geq f(x) = f(n(T)/T) = f(n(T))/T$. Since this inequality is true for any schedule we have $\hat{F} \geq F(T)/T$ and, by taking the limit, $\hat{F} \geq F$. ∎

Lemma 2: $\hat{F} \leq F$.

Proof: Given an optimal solution x of (3) we build a schedule in the following way. First of all we introduce for each machine m a dummy operation called *idle operation* whose purpose is to satisfy the capacity constraints in (3) with equality. There is some freedom in the choice of processing time for the idle operations, since only the product of frequency times the processing time is specified. So we may assume that $\sum_{a \in m} p_a x_a = 1$ for each machine.

Now whenever a machine has completed an operation the choice of the next operation to execute is based on the following rule: select the operation for which the ratio $n_a(t)/x_a$ is minimum at the current time t among the operations executable by the machine (i.e. $x_a > 0$). We are now going to show that, by this rule, $n_a(t)/t \to x_a$ as $t \to \infty$. Consequently $n_j(t)/t \to x_j$ and

$$\hat{F} = f(\{x_j\}) = f(\lim \left\{ \frac{n_j(t)}{t} \right\}) = \lim f(\left\{ \frac{n_j(t)}{t} \right\}) = \lim \frac{f(\{n_j(t)\})}{t} \leq \lim \frac{F(t)}{t} = F$$

In order to show that $n_a(t)/t \to x_a$, we first show that the inequality

$$\frac{n_h(t) - 1}{x_h} \leq \frac{n_k(t)}{x_k} \tag{4}$$

holds for any pair $h, k \in m$ of operations for any time t. In fact during the execution of the operation h, one has $n_h(t)/x_h \leq n_k(t)/x_k$ for any k due to the choice rule. At the instant when operation h is completed, $n_h(t)$ is increased by one and so (4) holds. Until operation h is executed again in a subsequent time the left hand side of (4) does not increase whereas the right hand side possibly does for some operation k. Therefore (4) holds for all instants of time, provided it holds at the initial time, which is true since $n_h(0) = 0$, $\forall h \in m$. If we define $a(t)$ to be the operation being executed at time t, clearly we have by the choice rule

$$\frac{n_{a(t)}(t)}{x_{a(t)}} \leq \frac{n_h(t)}{x_h} \quad \forall h, \forall t \tag{5}$$

Now (4) and (5) can be used to derive lower and upper bounds on the ratio $n_h(t)/t$. In order to derive a lower bound, let us multiply both sides of (4) by $p_h\, x_h$ yielding

$$p_h\, n_h(t) - p_h \leq p_h\, x_h \frac{n_k(t)}{x_k} \quad \forall h, \forall k, \forall t \tag{6}$$

Let us focus our attention on a generic operation k at a time t and consider separately the alternative cases when k is in execution at time t and when it is not. Let us first consider the latter case ($a(t) \neq k$) and sum (6) for $h \neq k$ and $h \neq a(t)$, plus the inequality

$$p_{a(t)}\, n_{a(t)}(t) \leq p_{a(t)}\, x_{a(t)}\, \frac{n_k(t)}{x_k} \tag{7}$$

derived from (5) and the identity

$$p_k\, n_k(t) = p_k\, x_k\, \frac{n_k(t)}{x_k} \tag{8}$$

Recalling that $\sum_{h\in m} p_h\, x_h = 1$, one gets

$$\sum_{h\in m} p_h\, n_h(t) - \sum_{\substack{h\neq k \\ h\neq a(t)}} p_h \leq \frac{n_k(t)}{x_k} \qquad k \neq a(t) \tag{9}$$

Clearly the following inequalities hold for any t (for the left-hand-side inequality recall that, including dummy operations, machines are never idle)

$$t - p_{a(t)} \leq \sum_{h\in m} n_h(t)\, p_h \leq t \tag{10}$$

By combining (9) and the left-hand-side of (10) one gets

$$t - \sum_{h\neq k} p_h \leq \frac{n_k(t)}{x_k} \tag{11}$$

holding for all operations k which are not in execution at time t. If the operation k is in execution at time t ($k = a(t)$) we sum (6) for $h \neq k$ plus the identity (8) to get

$$\sum_{h\in m} p_h\, n_h(t) - \sum_{h\neq k} p_h \leq \frac{n_k(t)}{x_k} \qquad k = a(t)$$

and by (10)

$$t - \sum_{h\in m} p_h \leq \frac{n_k(t)}{x_k} \tag{12}$$

Then the following inequality holds for any k and any t:

$$t - \sum_{h\in m} p_h \leq \frac{n_k(t)}{x_k}$$

i.e., by defining $P_m := \sum_{h\in m} p_h$ and $m(k)$ the machine used by operation k:

$$x_k\, (1 - \frac{P_{m(k)}}{t}) \leq \frac{n_k(t)}{t} \qquad \forall k,\ \forall t \tag{13}$$

In order to derive an upper bound let us rewrite (5) as

$$x_h\, \frac{n_{a(t)}(t)}{x_a(t)} \leq n_h(t),$$

multiply both sides by p_h and sum these inequalities for all h. Recalling again that $\sum_{h\in m} p_h\, x_h = 1$, one gets

$$\frac{n_{a(t)}(t)}{x_{a(t)}} \leq \sum_{h\in m} p_h\, n_h(t)$$

which, combined with the right-hand-side of (10), yields

$$\frac{n_{a(t)}(t)}{x_{a(t)}} \leq t \tag{14}$$

holding for all activities in execution. In order to deal with activities k not in execution, let t be a time at which k is not in execution and τ a time at which it was last in execution (if k was never executed before the upper bound is trivial). Then $n_k(\tau) = n_k(t) - 1$ and, by applying (14),

$$\frac{n_k(t) - 1}{x_k} = \frac{n_k(\tau)}{x_k} \leq \tau \leq t$$

$$n_k(t) \leq x_k\, t + 1 \qquad k \neq a(t) \tag{15}$$

By comparing (14) and (15) we see that (15) holds for any k and any t. Then we have

$$\frac{n_k(t)}{t} \leq x_k + \frac{1}{t} \tag{16}$$

and from (13) and (16) it is clear that $n_a(t)/t \to x_a$.

It remains to prove that the asymptotic schedule can be indeed carried out in the sense it always finds in the buffer the item necessary for the execution of the operation required by the choice rule. Let v be a node in the graph. We want to find an upperbound to the difference $\Delta_v^-(t)$ between the number of operations in v^+ already started within time t and the number of operations in v^- already completed within t. This difference provides a bound on the possibility for the operations to be out of stock. We further subdivide the operations in v^+ which have already started, into completed operations, denoted by v_c^+, and operations in execution, denoted by v_e^+. So we have

$$\Delta_v^-(t) = \sum_{a \in v_c^+} n_a(t) + \sum_{a \in v_e^+} (n_a(t) + 1) - \sum_{a \in v^-} n_a(t)$$

and therefore, by using (15), (14) and (13) to bound the three sums respectively we get

$$\Delta_v^-(t) \leq \sum_{a \in v^+} (t\, x_a + 1) - \sum_{a \in v^-} (t\, x_a - P_{m(a)}\, x_a)$$

from which, using the balance equation we finally get

$$\Delta_v^-(t) \leq |v^+| + \sum_{a \in v^-} P_{m(a)}\, x_a =: L_v \tag{17}$$

This bound does not depend on t and therefore if each buffer always has at least L_v items for each node v of the buffer no operation will be out of stock. In order to fill up the buffer there is only the need to run an initialization phase. After this phase the asymptotic schedule can be started.

A second question concerns the verification that there is a bound on the size of any buffer. The maximum number of items in the buffer is given by the difference $\Delta_v^+(t)$ between the number operations in v^- already completed within time t and the number of operations in v^+ already started within time t (it is just the opposite of the previous quantity). Using the previous notation this difference can be expressed as

$$\Delta_v^+(t) = \sum_{a \in v^-} n_a(t) - \sum_{a \in v_c^+} n_a(t) - \sum_{a \in v_e^+} (n_a(t) + 1) \tag{18}$$

and bounded as

$$\Delta_v^+(t) \leq \sum_{a \in v^-} n_a(t) - \sum_{a \in v^+} n_a(t) \tag{19}$$

Therefore by using (15) and (13) one gets from (19)

$$\Delta_v^+(t) \leq \sum_{a \in v^-} (x_a\, t + 1) - \sum_{a \in v^+} (x_a\, t - P_{m(a)}\, x_a) = |v^-| + \sum_{a \in v^+} P_{m(a)}\, x_a =: U_v \tag{20}$$

Taking into account that at least L_v items must be always present in node v, the buffer size which allows execution of the asymptotic schedule is $\sum_{v \in B} L_v + U_v$. ∎

It is interesting to note that, as a by-product of the previous proof, we obtain a way of implementing a constraint on the buffer size through the following linear inequalities

$$\sum_{v\in B} L_v + U_v = \sum_{v\in B}(|v| + \sum_{a\in v} P_{m(a)}\, x_a) \leq K_B \tag{21}$$

with K_B maximum buffer size for buffer B, $|v|$ the degree of node v and $a \in v$ meaning all operations associated to arcs incident with v.

We may wonder whether it is not too restrictive using in (21) the expression $\sum_{v\in B}(|v| + \sum_{a\in v} P_{m(a)}\, x_a)$ which is a bound on the needed buffer size rather than the needed buffer size itself. However, the bounds L_v and U_v can be considered tight, as we now show, and this justifies using the inequality (21). As for L_v, we present a simple example which actually shows that the three bounds used to establish (17) are separately tight for different, though arbitrarily close, values of t. The choice rule, simple in itself, gives rise however to complex and almost impredictable patterns for the step functions $n_a(t)$ so that it is safe to assume that there exist time instants at which all three bounds are tight.

Consider q operations on the same machine with same frequency $x_k = 1/q$ and processing times $p_1 = \varepsilon$ and $p_k = (q-\varepsilon)/(q-1)$, $k \neq 1$, with $\varepsilon > 0$ arbitrarily small. The operations are processed according to a rotation schedule in index order. Then we have for $t = h\,q + 2\,\varepsilon$ and h arbitrary integer

$$n_1(t) - x_1\, t - 1 = (1 + \left\lfloor \frac{t-\varepsilon}{q} \right\rfloor) - \frac{t}{q} - 1 = (1 + \left\lfloor \frac{h\,q+\varepsilon}{q} \right\rfloor) - \frac{h\,q+2\,\varepsilon}{q} - 1 = -\frac{2\,\varepsilon}{q}$$

which establishes that (15) is tight. Also (14) is tight if we consider the second operation (which at $t = h\,q+2\,\varepsilon$ is in execution). For $t = q\,h - \varepsilon$, with h integer we have

$$n_q(t) + x_q\,(P_m - t) = \left\lfloor \frac{t}{q} \right\rfloor + \frac{1}{q}\,(q-t) = \left\lfloor \frac{q\,h-\varepsilon}{q} \right\rfloor + 1 - \frac{q\,h-\varepsilon}{q} = \frac{\varepsilon}{q}$$

which establishes that (13) is tight. Now note that U_v is computed from (19) rather than the stricter relation (18). We may build an instance such that the pieces in a buffer are fed to the respective machines all at the same time. At that time there are no operations in execution, so (19) is tight and U_v is tight as well.

5. SET-UP TIMES AND BATCH SIZE

When a machine is devoted to process different pieces it may require a specific set-up time before starting processing a piece. In this case it is convenient to process in sequence several pieces in order to reduce the negative effect due to the set-up time. On the other side a long sequence of pieces requires a larger buffer size. In this section we address the problem of finding a proper balance between these two opposite requirements.

Given an operation a let δ_a be its set-up time, s_a be the frequency of these set-ups and L_a be its batch size. Let us note that sequence dependent set-up times cannot be accomodated within this model. The capacity constraints must be modified into:

$$\sum_{a\in m} x_a\, p_a + s_a\, \delta_a \leq 1 \tag{22}$$

and the set-up frequency is linked to the batch size through

$$x_a \leq s_a\, L_a \tag{23}$$

In order to have a meaningful model we have to take into account also the buffer size as expressed in (17) and (20) and in the constraints (21). Here we must consider the processing times of the whole batches, i.e.

$$\sum_{v\in B} |v| + \sum_{a\in v} (\sum_{h\in m(a)} L_h\, p_h\,)\, x_a \le K_B \tag{24}$$

The constraints (22), (23) and (24) may be used to assess a convenient batch size with respect to the required process performance.

6. A NUMERICAL EXAMPLE

Applying the previous results to a particular case is straigthforward in general so we limit ourselves to show one simple example. Let us consider Example 1 with 6 presses and processing times (in minutes) $p_1 = 8$, $p_2 = 8$, $p_3 = 8$, $p_4 = 6$, $p_5 = 6$, $p_6 = 5$ and with 3 AGV's with travel times $q_1 = 3$, $q_2 = 4$, $q_3 = 5$, $q_4 = 5$, $q_5 = 4$, $q_6 = 3$. The linear programming problem (3) particularized as in (2) (with a slight simplification because of redundant constraints) becomes

$$\begin{array}{lrcll} \max & x & & & \\ & p_h\ x_h & \le & 1 & h \in H \\ \sum_{h\in H} & q_h\ x_{hk} & \le & 1 & k \in K \\ \sum_{h\in H} & x_h & = & x & \\ & x_h & = & \sum_{k\in K} x_{hk} & h \in H \end{array}$$

yielding the following optimal solution (only non zero solutions are shown): $x = 0.7883333$ $(= 473/600)$,

$$x_1 = x_{11} = x_2 = .125 = 1/8, \quad x_{21} = .025 = 1/40, \quad x_{23} = .1 = 1/10, \quad x_3 = x_{32} = 1/15$$

$$x_4 = x_{41} = 21/200, \quad x_5 = x_{52} = 1/6, \quad x_6 = x_{63} = 1/5$$

The optimal value for x means that on the average every 1 min and 16.1 sec (=600/473 minutes) one item is brought to the depot. The optimal values for x_1, x_2, x_5, x_6 indicate that these presses work without interruption, whereas press 3 starts working on the average every 15 minutes (hence being idle for 7 minutes) press 4 starts working on the average every 9.52 minutes (hence being idle for 3.52 minutes). The AGV's work without interruption.

As for the press buffers we recall that each node $h \in H$ is a buffer. So we need to compute $L_h + U_h$ for each $h \in H$. We have $|h| = 4$ and $P_{m(a)} = \sum_h q_h$ for any AGV mission a. Then (21) becomes

$$L_h + U_h = |h| + p_h\, x_h + \sum_h q_h \sum_{k\in K} x_{hk} = 4 + p_h\, x_h + 24 \sum_{k\in K} x_{hk} \le K_B \qquad h \in H \tag{25}$$

The following six values are obtained for $L_h + U_h$: 4.04, 4.04, 2.173, 3.19, 5.04, 5.84. Supposing that buffers have a limited size of four items we add the inequality (25) (with $K_B = 4$) obtaining the new solution $x = 0.7552$,

$$x_1 = x_{11} = 0.12375, \quad x_2 = 0.12375, \quad x_{22} = 0.00575, \quad x_{23} = 0.118, \quad x_3 = x_{32} = 0.1134,$$
$$x_4 = x_{41} = 0.2575, \quad x_5 = x_{52} = 0.132, \quad x_6 = x_{62} = 0.1365$$

with buffers 1, 2, 5 and 6 used at full capacity.

Applegate, D., W. Cook (1991), "A computational study of job shop scheduling", *ORSA Journal on Computing*, **3**.

Ball, M.O., T.L. Magnanti, C.L. Monma and G.L. Nemhauser eds. (1995 a), *Handbooks in Operations Research and Management Science, 7: Network Models*, Elsevier Science, Amsterdam.

Ball, M.O., T.L. Magnanti, C.L. Monma and G.L. Nemhauser eds. (1995 b), *Handbooks in Operations Research and Management Science, 8: Network Routing*, Elsevier Science, Amsterdam.

Garey, M.R., D.S. Johnson (1979), *Computers and intractability: a guide to the theory of NP-completeness*, W.H. Freeman and Company, San Francisco, CA, USA.

Graves, S.C., A.H.G. Rinnooy Kan and P.H. Zipkin eds. (1993), *Handbooks in Operations Research and Management Science, 4: Logistics of Production and Inventory*, Elsevier Science, Amsterdam.

Papadimitriou, C., P. Serafini and M. Yannakakis (1993), "Computing the throughput of a network with dedicated lines", *Discrete and Applied Mathematics*, **42**, p. 271-278..

Pochet, Y. (2001), "Mathematical programming models and formulations for deterministic production planning problems", in *Computational combinatorial optimization (Schloß Dagstuhl, 2000)*, M. Jünger and D. Naddef eds., Springer, Berlin.

Silver, E.A., D.F. Pyke and R. Peterson (1998), *Inventory Management and Production Planning and Scheduling*, Wiley and Sons, New York.