International Journal of Production Research
Vol. 00, No. 00, 00 Month 200x, 1–21

Coordinated scheduling of a single machine with sequence dependent setup times and time window constraints

Payman Jula[a*] and Arash Rafiey[b]

[a]*Faculty of Business Adminstration, Simon Fraser University, Burnaby, B.C., Canada*; [b]*School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada*

(*June 2010*)

In this paper we consider selecting and scheduling of several jobs on a single machine with sequence dependent setup times and strictly enforced time windows constraints on the start-times of each job. We demonstrate how to develop network-based algorithms to sustain the desired work in process (WIP) profile in a manufacturing environment. Short-term production targets are used to coordinate decentralized local schedulers and to make the objectives of specific areas inline with the chain objectives. Wide range of test problems with two different network structures are simulated. The effectiveness, efficiency, and robustness of the proposed algorithms are analyzed and compared with an exhaustive search approach.

Keywords: Machine scheduling; time windows constraints; sequence dependent setups; manufacturing; supply chain coordination

*Corresponding author. Email: pjula@sfu.ca

ISSN: 0020-7543 print/ISSN 1366-588X online
© 200x Taylor & Francis
DOI: 10.1080/00207540xxxxxxxxx
http://www.informaworld.com

1. Introduction

Motivated by scheduling challenges in complex manufacturing systems, this article addresses the job selections and scheduling problem on a single machine, considering sequence dependent setup times while observing the time window constraints associated with each job. The goal is to maintain an appropriate work in process (WIP) profile in the system which is designed to satisfy several supply chain manufacturing objectives while observing many constraints.

Many steps of advanced manufacturing systems have a time window constraint associated with each job, dictating the earliest and the latest time that consecutive step can start. For instance in hot metal rolling industries, where the heated metal has to undergo a series of operations at continuously high temperatures before it is cooled in order to prevent defects. Similarly, in the plastic molding and silverware production industries, a series of operations must be performed to immediately follow one another to prevent degradation.

As part of the process flow in semiconductor manufacturing, wafers go through series of etch and diffusion. Etch is a process where wafers get exposed to series of chemicals to remove particles from their surfaces. To achieve quality products, the etched wafers have to go through the diffusion steps within strictly defined time where layers are deposited on wafers. Within the sequence of deposition steps, the surface of some layers is unstable and top layer must be deposited within a strictly defined time interval of the previous step to prevent the wafer to be reworked or scraped. There are many other process steps in semiconductor manufacturing which dictates a time window for the start time to prevent the corrosion (e.g. in metal layers) or contamination.

In addition to the time window constraints, there are many process steps which require setups with setup times depending on previous state of the tool. For example, as part of semiconductor manufacturing process flow, wafers go through coating in preparation for the lithography in which small features are printed on the wafer. For better quality, photo engineers often dictate a time window that wafers must follow to go through the lithography step after coating. The earliest time is observed to give the coat enough time to settle and the latest time is required to avoid contamination. Based on its previous job type or machine state, the lithography tool may need some time for setup for the new type of job. This time is required for the machine to get an appropriate reticle and download appropriate software. Another example is the sort area in semiconductor manufacturing, where the wafers are probed by testers to identify the defected products and to guarantee the quality of products shipped to customers. Probing jobs have ready times and due dates associated with them. Depending on previous state, testers may need setup time to get appropriate type of load board and probe card and to reach to appropriate temperature, which may be different from the previous job's requirements. Resource availabilities are among other factors which may impose time window constraints. For instance, staff or tools availabilities may require jobs to be started or processed within a specific time frame.

The characteristics of resources and constraints make this scheduling problem complicated and challenging. Furthermore, job arrival rates, availability of resources and management priorities may change over time. A flexible scheduling system is required to adapt and react efficiently, effectively and robustly in such a dynamic environment. In this context, an *effective* schedule is defined to be a schedule that attains desired targets. An *efficient* schedule is a schedule that can be easily generated and is applicable to industrial settings. A *robust* schedule is the one that performs well in most given situations,

even in the presence of significant uncertainties.

Traditionally, schedulers have been designed to address local performance objectives such as maximizing the on-time delivery at the step, minimizing tardiness, or minimizing makespan at the step. The above local objectives are not necessarily in line with the chain objectives. Local schedulers should be coordinated to achieve supply chain goals. In this paper, we follow the coordinating framework suggested by Jula and Leachman (2008). In this framework, the desired performance of the chain is translated into production shift (simply called shift in this document) targets at each step to maintain an appropriate WIP profile throughout the system. This article is the first that discusses the details of developing schedulers with sequence-dependent setups and time-windows constraints within this framework.

The purpose of this paper is to investigate methods for improving the scheduling of several jobs on a single machine with time constraints considering sequence dependent setup time while observing a pre-specified time horizon. The contributions of this paper are in several domains: a) it introduces a scheduling methodology that uses short-term production targets to maintain a target WIP profile at each step. In this framework, decentralized local schedulers are coordinated towards achieving the supply chain manufacturing goals, b) it explicitly considers the time windows associated with each job which indicates the earliest and the latest time that job should be started, c) it addresses the complexities of sequence dependent setup times. To the best of our knowledge, this paper is the first to address the coordinated sequencing of jobs with time windows constraints observing sequence dependent setup times in manufacturing environments. The paper studies the efficiency, effectiveness, and robustness of the proposed algorithms.

The rest of the paper is organized as follows: in Section 2, the previous relevant literature is reviewed. In Section 3, the problem is described. Several algorithms are developed and explained in Section 4. The results of sets of experiments are reported and analyzed in Section 5. Finally, conclusions, recommendations and directions for future research are presented in the last section.

2. Related literature

The problem of scheduling single machine with sequence dependent setup time can be translated to Traveling Salesperson Problem (TSP), in which each city corresponds to a job and the distance between cities corresponds to the time required to change from one job to another. There are many publications in the domain of TSP and scheduling single machine with sequence dependent setup times (Allahverdi *et al.*, 1999, 2008). Zhu and Wilhelm (2006) provide a literature review on scheduling and lot sizing with sequence dependent setup time and cost.

Imposing time constraints in the system significantly complicate the problem. Lenstra *et al.* (1977) show that minimizing the total weighted completion time on a single machine with job ready time or job due dates is an NP-hard problem. Even finding a feasible solution to the sequencing of jobs on a single machine to satisfy required time windows is an NP-complete problem (Garey and Johnson, 1979).

Most of the researchers have either considered the job release time or job due dates, but not both constraints. In a deterministic environment, the job due dates may be translated to the latest time that job can be started. Minimizing the weighted completion time on a single machine subject to due dates has been studied by several authors (e.g. Smith 1956, Potts and Wassenhove 1983, Posner 1985). Ahmadi and Bagchi (1986), Chand

and Schneeberger (1988) investigate enumerative procedures to minimize total weighted earliness of jobs on a single machine with no wait time and pre specified due dates. The no wait time process flows (both with due dates and without due dates) are also special cases of the problem addressed in this article. For recent works on no-wait time scheduling, see, for example, Li *et al.*(2008) and Ruiz and Allahverdi (2009). Many authors (e.g. Bianco and Ricciardelli 1982, Hariri and Potts 1983, and Belouadah *et al.* 1992) have examined the problem of minimizing the total weighted completion time with job release date on a single machine. None of the above articles consider both release times and deadlines for jobs.

Ahmadi and Bagchi (1992) have considered both release date and due dates constraints in a flow shop problem. However, they assume non-constraining release times in their solution approach. Gélinas and Soumis (1997) use dynamic programming to minimize the total weighted completion time considering both job release times and due dates. Pan and Shi (2005) propose a branch-and-bound algorithm for the problem under dual constraints. Kedad-Sidhoum and Sourd's (2010) article is among several articles that address one machine earliness-tardiness scheduling problem with penalties on earliness and tardiness of jobs. None of these articles which consider both the release date and the due dates have addressed the sequence dependent setup times.

Sourd (2006) addresses one machine scheduling problem with sequence dependent setup time and earliness-tardiness penalties considering release dates and due dates which are not hard constraints. Asano and Ohta (1996) consider the ready time and due-date as constraints for each job and also consider sequence dependent setup time between jobs. The authors propose a branch and bound approach to minimize the total tardiness of jobs. The computational time of their approach increases exponentially with the increase of the number of jobs and is not suitable for medium or large problem sizes (i.e. more than 30 jobs).

Another relevant body of work is the Traveling Salesperson Problem with Time Windows (TSPTW) in which a salesperson, initially located at the origin, must serve a number of geographically dispersed customers such that each customer is served within a specified time window. The objective is to find the optimum route with minimum total cost (time) of travel. TSPTW and its applications has been studied by many authors (e.g. Dumas *et al.* 1995, Carlton and Barnes 1996, Gendreau *et al.* 1998, Ascheuer *et al.* 2001). Asymmetric TSPTW is a special case of the problem under study in this article, in which the weights of jobs are equal, and there is no families defined for jobs. As will be discussed in the next section, assigning weights (scores) to jobs and having families of jobs are two essential elements in our coordinated scheduling approach in this article. Furthermore, the concept of having production targets for each family hasn't been addressed in TSPTW related articles.

Traditionally, schedulers have been designed to address local performance objectives that are not necessarily in line with the chain objectives. Allahverdi *et al.* (2008) identify 16 performance objectives that are used in more than 300 recent papers in the domain of scheduling problems with setup times or cost. Examples of these performance measures are: makespan, total earliness at the step, total setup/change over cost of the step, and total weighted flow time at the step. None of the identified performance objectives are directly linked to the performance of the chain.

It is important that the objectives of the local schedulers should be set so that the schedulers are coordinated with each other in order to achieve supply chain objectives. There is still a gap in the body of existing literature to address the design of local decentralized schedulers such that their performance become inline with the global chain

performance. This article targets this gap.

3. Background and problem statement

In this paper, our approach is to manage the supply chain manufacturing by maintaining a carefully designed dynamic WIP profile at each step of the system. We monitor the chain by monitoring the WIP status and control the system by trying to achieve a desirable WIP profile. The WIP profile is set according to managerial needs, which addresses combinations of objectives such as maintaining low WIP and cycle time while having high throughput. Other considerations such as bottleneck starvation avoidance, or long-unreliable transportation times can also be incorporated in the WIP targets. To achieve the desirable WIP profile, production targets are set dynamically for each shift for each product at each step of the chain. These targets can be used to evaluate the chain performance and guide decentralized schedulers to control the system so as to achieve desirable outputs. This framework also assists local schedulers to respond effectively and efficiently to the uncertainties that arise in dynamic environments.

3.1. *WIP management and scheduling*

Jula and Leachman (2008) established a framework to coordinate local decentralized schedulers by using the short term production targets. They have shown how these targets can be set in complex supply chain manufacturing. They derived, from high-level production plans, short-term production targets for use in decentralized low level scheduling. By this approach, they established a closed loop between higher-level production planners and local decentralized schedulers. Target-generating algorithms use long-term production targets to set the short term targets for the local scheduler. The performance of the schedulers affects the available work in processes, which in turn affects the production planner's results. The target generating unit is adaptive and reactive to the changes in the system and the performance of schedulers. Changes in the system and the performance of scheduler affect the WIP status, which in turn affects the future targets set by the target generator. Efficient local schedulers will then use the short-term production targets to control the processes.

To measure the earliness or lateness of a job at a station, information about the down stream WIP, yield, and final demands are required. Two parameters are used to specify the short term production targets, i.e. Ideal Production Quantity (IPQ), and Schedule Score (SS). The Ideal Production Quantity (IPQ) indicates the number of units of a device/step to be completed by the end of a shift in order to meet the target flow time and catch up with the supply chain manufacturing out schedule. The terms are adjusted by the planned yields from the step to chain out, so that all terms are expressed in units of production of the step under study. The IPQ can be expressed as:

IPQ = (The total supply chain outs due until one shift after the target cycle time from the step to supply chain out) - (actual supply chain output to date) - (the projected supply chain output from actual downstream WIP)

The amount is called "ideal" since it may be infeasible to process this amount during the current shift for a variety of reasons. There might not be enough WIP supplied to the step during the shift or there might not be enough qualified resources available to complete the IPQ in one shift.

The Schedule Score (SS), on the other hand, indicates the earliness or lateness of current production of a specific product at a step. The schedule score is presented based on the number of shifts that a particular product/step is ahead of the schedule or late in schedule if no WIP for that step is completed this shift.

$$SS = IPQ \ / \ (Average\ supply\ chain\ out\ rate)$$

Leachman *et al.* (2002) demonstrated the calculations of IPQs and SSs for a simple case. In complex situations where the products go through many branching, binning and substitutions, before reaching customers - calculating IPQs and SSs requires special algorithms which have been addressed by Jula and Leachman (2008). Jula and Leachman (2010) show how IPQs and SSs can be used to develop coordinated schedulers for parallel non-homogenous batch processing machines under multi resource constraints.

3.2. *Problem statement*

Let $\{J_1, J_2, ..., J_n\}$ be the set of non-identical jobs to be scheduled on a single machine. Each job J_i has a processing time t_i, schedule score (SS_i), and an associated time window $[a_i, b_i]$, in which a_i is the earliest possible starting time and b_i is the latest possible starting time for the job. There is a required setup time $s_{i,j}$ to execute job J_j after J_i. Readers should note that $s_{i,j}$ maybe different from $s_{j,i}$. Each job belongs to a family of jobs with a common pre-specified Schedule Score, and Ideal Production Quantity for a scheduling horizon (T). In this article we use production working shifts for the scheduling horizon.

There is no arrival of new jobs during the scheduling horizon. Machine can process one job at a time and when the process of a job is started on the machine, the process is not interrupted. Machine break-down is not considered in this article.

The goal is to select a sub-set of the jobs and schedule them on the machine during the scheduling horizon. The primary objective is to maximize the overall schedule score of scheduled jobs, while satisfying the target production for each family of jobs as much as possible. The scheduled jobs are credited up to the target production level of their families (IPQs). The Secondary objective is to minimize the maximum completion time of all jobs. The constraint is that the jobs are only allowed to be assigned to the machine during their time-windows.

4. Proposed solutions

We model the problem using network flow architectures, and provide our solutions using network flow concepts. We will first study the feasibility of scheduling all the available jobs on the machine in section 4.1. The details of construction of the network model that are used in subsequent algorithms is explained in section 4.2. We will then develop our solution in section 4.3 for a base case, by ignoring the family of jobs and their target productions, and under the assumption that the system temporal parameters are integer values of a basic time grid. This assumption is later relaxed in section 4.4 and a solution for the general case is provided. We then extend our approach to consider job families with target productions for each family in section 4.5.

4.1. *Feasibility analysis*

Before we propose our solutions, it is useful to discuss whether it is feasible to execute all the available jobs on the single machine. Readers should note that if the score of every job is set to one, then maximizing the overall score is equivalent to finding the maximum number of jobs that can be executed on the machine.

Consider two jobs J_i, and J_j, where $a_i \leq a_j$. We define J_i has *conflict* with J_j if $a_i + t_i + s_{i,j} > b_j$. If J_i has conflict with J_j, then we can not execute J_j after the execution of J_i. Please note that in this case J_j doesn't necessarily have conflict with J_i.

Problem 4.1: *Given n jobs and their associated time windows, is it feasible to schedule all jobs on a single machine?*

Solution: Let's set the score of each job to one. We then construct the vertex set G consisting of nodes $\{1, 2, .., n\}$, and arcs connecting node i to j where J_i has conflict with J_j.

Lemma 4.2: *If G has a cycle, then it is not feasible to execute all the jobs on a single machine, while satisfying their associated time windows constraints.*

Proof: Let $C = v_1, v_2, ..., v_m, v_1$ be a cycle in G. Without loss of generality assume that v_1 is executed first. Since there is an arc from v_1 to v_2 this implies that v_2 should be executed before v_1, similarly v_3 must be executed before v_2 and v_1, and hence v_n must executed before $v_1, v_2, ..., v_{n-1}$. However, arc $v_n v_1$ implies that v_1 must be executed before v_1, which is a contradiction. ◇

By the above Lemma, if G has a directed cycle then it is not feasible to schedule all jobs on a single machine. In this case, we may need extra capacity to be able to schedule all the jobs. Alternatively, higher-level production planner may be designed such that it assigns the jobs to different scheduling horizons on the machine.

4.2. *Network construction*

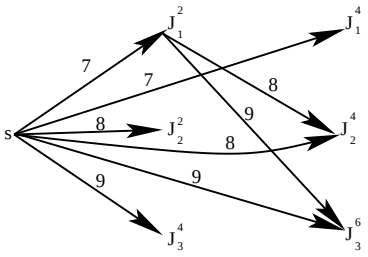
In this section the details of the construction of the network model that are used in subsequent algorithms is explained. Let's consider α to be a constant basic time grid length, which can be set according to the temporal characteristics of the system and the required accuracy of the results. Each job can be started only on time grid points.

There are different ways to construct the network based on this basic time grid. In this section, we propose the construction of the graph based on the Common Grid Points Network for all the jobs, called CGPN in this article. Later in Section 5, we will propose another method of constructing the network.

For simplicity let's assume α is reasonably small such that $b_i - a_i \geq \alpha$ (we will later on discuss the cases where this condition doesn't hold). Let $\gamma_i = \lceil a_i \rceil$, the next closest upper grid for a_i. We construct the auxiliary digraph D as follows: for each job J_i consider a set vertices $J_{\gamma_i}, J_{\gamma_i+\alpha}, ..., J_{\gamma_i+t\alpha}$, where $\gamma_i + t\alpha \leq b_i$ and t is a positive integer number. Vertex $J_{\gamma_i+t\alpha}$ represents a possible starting time of job J_i, and $\gamma_i + t\alpha$ is the time associated with this vertex. Consider vertex s as the first time when machine becomes available, and without loss of generality, let's associate time zero to this vertex. We then add an arc from vertex $u = J_{\gamma_i+t\alpha}$ to vertex $v = J_{\gamma_j+t'\alpha}$ if these conditions hold:

- $i \neq j$
- $\gamma_j - \gamma_i + (t' - t)\alpha$ is not less than $t_i + s_{ij}$.

We add an arc from the start node s to u with the weight of SS_i. The weight of the



(a) Network D

	J_1	J_2	J_3
J_1	0	0	1
J_2	2	0	2
J_3	2	0	0

(b) Setup time

	a	*b*	Process time	Score
Job J_1	1	4	2	7
Job J_2	2	5	3	8
Job J_3	3	6	3	9

(c) Time windows, process times, and schedule scores

Figure 1. Network diagram.

arc uv, $e(uv)$, is then set to be the schedule score of J_j, $e(uv) = SS_j$.

Let P be a path from s to vertex x. Then, there is a sequence of jobs on path P. Node $u = J_{\gamma_i+t\alpha} \in P$ means that job J_i belongs to this sequence and should be started at time $\gamma_i + t\alpha$. The sum of the weights of the arcs over P is then the total schedule score for this schedule.

Figure 1.a. shows an example of such a network constructed for a simple case of three jobs. In this figure, the basic time grid $\alpha = 2$, and J_i^t denotes vertex $J_{\gamma_i+t\alpha}$, which indicates job i starts at time $\gamma_i + t\alpha$. For example, J_2^4 is associated with the vertex which indicates the possible start time of the second job is at the time 4. The index $[i, j]$ of the setup time table depicted in Figure 1.b. corresponds to sequence dependent setup time $s_{i,j}$. Therefore, for example, it takes one unit of time in order to execute job 3 after job 1. Figure 1.c. shows the time windows, process time, and schedule score associated with each job. In this network, for example, J_1^2 is connected to J_2^4 because the total of start time and processing time for job 1 and the setup time for job 2 (2+2+0=4) is not greater than the possible start time of job 2 at time 4.

4.3. *Base case analysis*

We develop our solution in this section for a base case, in which the family of jobs are ignored and there is no production targets for the jobs. Furthermore, here, we assume that the system temporal parameters are integer values of a basic time grid.

Problem 4.3: *Given a single machine and n jobs, find the best set of jobs and their sequence such that the overall schedule score is maximized (primary objective), while the maximum completion time is minimized (secondary objective).*

Here is an explanation of our proposed algorithm to find the best schedule on a single machine case stated in problem 4.3: let $W(u)$ denote the weight of the maximum path from s to u and let $P(u)$ denotes the last vertex before u on the maximum path from s to u. At the beginning of the algorithm, $W(u)$ is set to be the weight of arc $e(su)$. At each step of the algorithm we choose a vertex u which a) has not been chosen before,

and b) u has in-degree zero from the unchosen vertices (i.e. there is no edge entering u from unchosen vertices), and c) $W(u)$ is maximum. For every v from unchosen vertices connected to u with the edge value of $e(uv)$, $W(v)$ is then updated to be equal to $W(u)+e(uv)$, and $P(v)$ is set to be equal to u, if the followings hold:

- $W(v) < W(u)+e(uv)$,
- there is no vertex v' which corresponds to job J_j on the maximum path from s to u.

Finally, the generated solution need to be modified to minimize the completion time. Suppose the last job on P is J_n which starts on $\gamma_n + t\alpha$. We then delete all the vertices of D where their associated time is not less than the associated time to the last vertex on P, i.e. $\gamma_j + t'\alpha \geq \gamma_n + t\alpha$. We try to find the maximum path P' in the remaining digraph. If the overall weight of P' is less than the overall weight of P then the algorithm stops, otherwise it repeats. Here is the detail of the algorithm:

Algorithm 4.4:

(1) $FV = (s)$, $fw = 0$, $Ch = \{s\}$, *and* $V(D)$ = *set of all vertices in* D;
(2) For every vertex v, *set* $W(v) = e(sv)$;
(3) Repeat $(a), (b), (c)$, N *times (*N *is the number of nodes in the network):*
  a) choose vertex $u \notin Ch$, *where* $W(u)$ *is maximum, and* u *has in-degree zero in* $V(D)\backslash Ch$;
  b) for every vertex $v \in V(D)\backslash Ch$, *where* uv *is an arc, if:*
    i. $W(u) + e(uv) > W(v)$,
    ii. there is no any other vertex v' *on the maximum path from* s *to* v, *which both* v' *and* v *correspond to the same job,*
  then set $W(v) = W(u) + e(uv)$, *and* $P(v) = u$;
  c) add u *to* Ch;
(4) Find vertex x *where* $W(x)$ *is maximum;*
(5) Consider an empty vector F, *and set* $i = 1$, *and* $y = x$;
(6) While $(y \neq s)$
  a) $F(i) = y$, $y = P(y)$, $i = i + 1$;
(7) If $W(x) \geq fw$, *then set* $fw = W(x)$ *and* $FV = F$;
(8) Let x *be* $J_{\gamma_i + t\alpha}$ *for some* i, *and* t; *delete all vertices* $J_{\gamma_j + t'\alpha}$ *with* $\gamma_j + t'\alpha \geq \gamma_i + t\alpha$ *from* D, *and go to step 2;*
(9) Output fw *and the reverse of* FV. ◇

4.3.1. Time Complexity (Efficiency)

In general, if the basic time grid (α) is large, then the chance that we miss some of the optimum solutions increases. On the other hand, if α is small then the number of vertices in D increases and the time complexity of algorithms grows, but depending on the granularity of the temporal parameters of the system, the algorithms may generate better results in term of our objective.

Suppose for every $1 \leq i \leq n$, $(b_i - a_i)/\alpha \leq r$. Then there are maximum nr nodes in the digraph D. The complexity of finding the maximum path in D is in the order of $O(r^2n^2)$. More precisely the complexity is on order of $O(m)$ where m is the number of arcs in D and m is at most r^2n^2. At the end of the algorithm, we delete some of the nodes and repeat the whole process again. Finding maximum weight path is repeated at most b_n/α times, where b_n is the last possible start time of the last job. Since $b_n \leq T$, then the time complexity of the Algorithm 4.4 is $O(r^2n^2T/\alpha)$, where T is the scheduling horizon.

4.3.2. Effectiveness

Lemma 4.5: *If the temporal parameters $(a_i, b_i, t_i, s_{i,j})$ for each job J_i are multiples of α, then Algorithm 4.4 finds the optimal solution.*

Proof: We first show the following claim:

Claim 4.6: *Let D be an acyclic digraph in which each vertex has a weight. Then, Algorithm 4.4 finds the maximum weighted path in D.*

Digraph D has a vertex (source) which is adjacent to all other vertices. At each step, the algorithm picks a vertex v with no in-neighbor among all the unselected vertices. The maximum weighted path from the source to v is the weight of v plus the weight of maximum path from source to u where uv is an arc in D, and u has the maximum weighted path among all the in-neighbors of v. It is clear that this algorithm finds the maximum weighted path.

Now suppose each job can start only at one particular time, i.e. $a_i = b_i$ for all job J_i. Then, finding the optimal solution for the jobs translates to finding the maximum weighted path in the corresponding network, which is done by the algorithm. However, in general $a_i \neq b_i$, therefore, there maybe several vertices for job J_i in the graph. Since all $a_i, t_i, s_{i,j}$'s are integer multiples of the basic time grid (α), then corresponding to any start time of each job J_i, there exists a node in the network. The algorithm then finds the maximum weighted path P from the source to the vertex v; corresponding to some starting point of job J_i. Moreover, P does not contain any other vertex u which corresponds to the same job J_i. Thus algorithm finds the optimal solution.

Consider an optimal solution in which some of the jobs do not start at integer multiples times of α. Here, we show that there exists an equal optimal solution in which all jobs start at multiple points of α. Let J_i be the first job in a sequence of jobs in an optimal solution in which jobs doesn't start at integer multiples of α. If $i > 1$, then J_{i-1} starts at multiple of α and since the process time of J_{i-1} is multiple of α, and also setup time between J_{i-1}, J_i is multiple of α, then J_{i-1} ends at a multiple point of α. Since J_i could start at multiple of α, we can start J_i earlier such that its start point becomes multiple of α. Note that this is possible because the time interval of J_i is at least α. Therefore, if the first job does not start at multiple of α then we can shift back and start it at a multiple point of α. Hence, we may assume that in optimal solution all jobs start at multiples of α. ◇

In most industrial environments, a basic time grid can be suggested such that the temporal parameters of the system can be expressed based on this time grid. In semiconductor manufacturing, for example, the processing times and time intervals for a product or batch of products are usually expressed in minutes. Therefore, having the basic time grid of one minute is sufficient to generate an acceptable and executable optimal solution.

The problem arises when temporal paraments of the system such as t_i or a_i is not integer value of the basic time grid for some job J_i. In this case the proposed algorithm may generate a sub-optimal solution. In the following lemma, we discuss some properties of a sub-optimal case.

Lemma 4.7: *Let SO be the overall schedule score of the solution obtained by Algorithm 4.4, and OPT be that of the optimal solution with the following conditions:*

- *a_i, b_i are multiples of α, and $b_i - a_i \geq \alpha$;*
- *processing time of each job $t_i \geq \alpha$ (but not necessarily multiples of α) ;*

Then, $SO \geq \frac{OPT}{2}$.

Proof: If all the jobs in OPT have starting time which are multiples of α then correspond to these jobs there are vertices in the network constructed in the algorithm and hence $SO = OPT$. So, suppose for some J_i in OPT the starting time is not a multiple of α. Let $St_1, St_2, ..., St_m$ be the starting time of the jobs in OPT. Consider two jobs J_{i+1} and J_{i-1} in OPT. If $SS_{i+1} < SS_i$ then by deleting J_{i+1} and shifting J_i to $\lceil St_{i+1} \rceil$ we can find a vertex correspond to J_i in the network. Similarly if $SS_{i-1} < SS_i$ then by deleting J_{i-1} and shifting J_i to $\lfloor St_{i+1} \rfloor$ we can find a vertex correspond to J_i in the network. This way by deleting at most half of the jobs in OPT we can find the corresponding vertices in the network constructed for the rest of the jobs in OPT; thus $SO \geq \frac{OPT}{2}$. $\diamond$

Considering above lemma, and based on our study, in the following section, we propose an enhancement to Algorithm 4.4.

4.4. *Shift-based algorithm*

Please note that based on the network structure, jobs can start only on time grid points. Therefore, depending on the granularity of α, the algorithm may impose idle times, and therefore the solution may need to be refined to accommodate more jobs. In this section, we propose a new shift-based algorithm that enhances solutions produced by Algorithm 4.4. This new algorithm moves some of the jobs in order to add more jobs that might not been scheduled by the previous algorithm due to the granularity of time grids. Suppose S is the set of jobs found by Algorithm 4.4. Then, shift-based algorithm finds the best solution among all solutions which contains set S as a sub-solution.

Here is a brief explanation on how this algorithm works. Let P be the maximum path found by the Algorithm 4.4. Note that there is no directed cycle in D, because the network is constructed such that there is no arc in which the start time of its beginning is greater than the start time of its end point. Let $K_1, K_2, ..., K_m$ be the jobs in P. Consider the machine Gantt chart; We define the gap G_r to represent the gap between the finish time of job K_r and the start time of job K_{r+1}. This gap can be expressed as $\gamma_{r+1} + t\alpha - \Gamma_r$, where Γ_r is the completion time of job K_r in P. Note that these gaps are ordered set where G_r is before G_{r+1}.

We say gaps $G_i, G_{i+1}, ..., G_{i+j}$ can be attached together if by shifting these gaps to the right (i.e. start jobs earlier) or to the left (i.e. start jobs later), or by shifting $G_1, G_2, ..., G_l$ to the left and $G_{l+1}, G_{l+2}, ..., G_{i+j}$ to the right, more jobs can be added into P. This interval is constructed only if it does not conflict with the time windows limits of the jobs in P. An auxiliary graph G can then be constructed from G_r's as follows. For each set of consecutive gaps $G_i, G_{i+1}, ..., G_{i+j}$ which can be attached together, an interval $[i, i+j]^*$ is introduced. This interval starts at gap i and ends at $i+j$, and $*$ is either $+$, $-$, or $\pm l$. The $+$ sign means that gaps $G_i, G_{i+1}, ..., G_{i+j}$ can be shifted to the right and $-$ means they can be shifted to the left, and finally $\pm l$ means that gaps $G_1, G_2, ..., G_l$ are shifted to the left and $G_{l+1}, G_{l+2}, ..., G_{i+j}$ are shifted to the right.

Job J_r can then be associated with this interval, if its time windows, and sequence dependent setup times (to the previous job and the next job) is consistent with this interval. For each interval, the schedule score of job J_r (SS_r) is then considered and for each interval $[i, i+j]^*$, a vertex with the same score as job J_r is then introduced. This means that if we choose to shift jobs $K_i, K_{i+1}, ..., K_{i+j}$ to the right (left) we can add job J_r, and therefore the total score of jobs increases by SS_r. An edge is then added between two vertices v_r, and v_s, if their corresponding intervals intersects. It is clear that G is an interval graph. For more information on interval graphs, interested readers are

encouraged to refer to (Golumbic, 1980).

Vertices of G correspond to some of the jobs that are not present in P. We then choose some of these vertices to maximize the overall score. Two vertices which are adjacent in G can not be selected, because their intervals intersect. Therefore, a maximum weight independent set I in G should be found. There is a polynomial time algorithm to find this independent set, see (Hsiao *et al.* 1992). Once set I is found, the corresponding jobs in I are then added to schedule P.

Figure 2 illustrates this procedure for an example of 7 jobs. Here, the scheduled jobs (1-4) are presented by cross-hatched areas on the machine's Gantt chart, while the remaining un-scheduled jobs (5-7) are shown by dark solid areas. The time windows of jobs $K_1, K_2, K_3, K_4, J_5, J_6, J_7$ are given by the following array:

$$\begin{bmatrix} 0 & 2 \\ 2 & 4 \\ 5 & 7 \\ 8 & 11 \\ 4 & 7 \\ 8 & 10 \\ 2 & 5 \end{bmatrix}$$

the processing times for jobs are $t = [2.5, 2.5, 2.5, 2.5, 1.5, 1, 1.5]$, and the schedule scores are $SS = [5, 5, 5, 5, 4, 3, 2]$. For simplicity, we do not consider setup times in this example (i.e. $s_{i,j} = 0$).

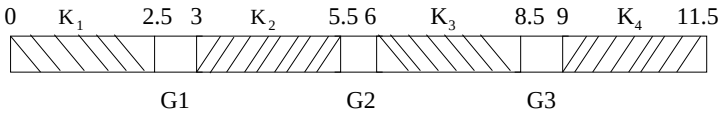
As shown in Figure 2.a., there are three gaps G_1, G_2, G_3 between the scheduled four jobs K_1, K_2, K_3, K_4. In the first part of the figure 2.b we move job K_2 to the left and shift K_3 to the right, so we obtain a gap of length 1.5 which can be used to schedule job J_5 with $SS = 4$. In the second part we move K_3 to the left and find a gap of length 1, which allow us to execute job J_6. Finally we shift jobs K_2, K_3 to the right and we find a gap of size 1.5 after job K_1, so we can schedule J_7 in this gap.

In Figure 2.c. we associate J_7 to the gaps $1, 2, 3$ since we were able to combine three gaps and process J_7 in the gap provided. Similarly, we associate J_5 to $1, 2, 3$ and finally J_6 is associated to $2, 3$. Figure 2.d. shows the interval presentation of jobs J_5, J_6, J_7. The procedure identifies J_5 as the best job that can be added to the schedule. Here is the details of shift-based algorithm.

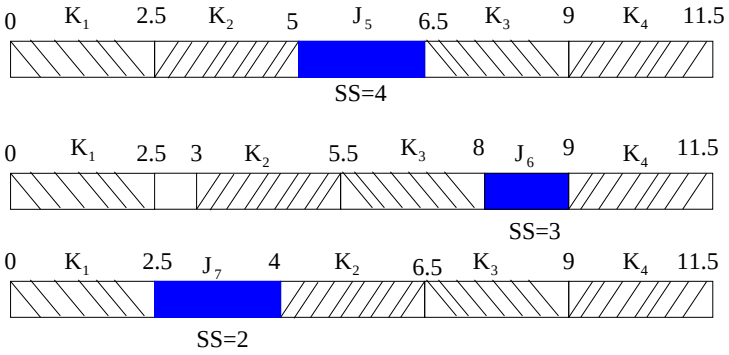
Algorithm 4.8:

(1) $FV = (s)$, $fw = 0$, $Ch = \{s\}$, *and* $V(D)$ *= set of all vertices in* D*;*
(2) For every vertex v*, set* $W(v) = e(sv)$*;*
(3) Repeats $(a), (b), (c)$*,* N *times (*N *is the number of nodes in the network):*
  a) choose vertex $u \notin Ch$*, where* $W(u)$ *is maximum, and* u *has in-degree zero in* $V(D)\backslash Ch$*;*
  b) for every vertex $v \in V(D)\backslash Ch$*, where* uv *is an arc, if:*
    i. $W(u) + e(uv) > W(v)$*,*
    ii. there is no any other vertex v' *on the maximum path from* s *to* v*, which both* v' *and* v *correspond to the same job,*
  then set $W(v) = W(u) + e(uv)$*, and* $P(v) = u$*;*
  c) add u *to* Ch*;*
(4) Find vertex x *where* $W(x)$ *is maximum;*

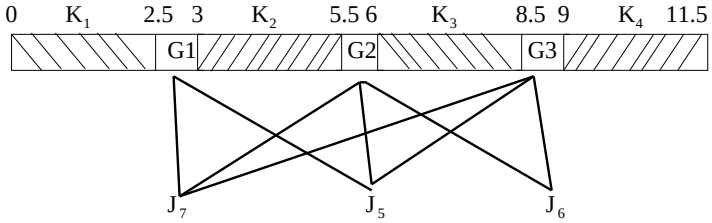
(a) Solution at the first stage of the algorithm



(b) Possible shifts



(c) Dependency graph



(d) Final interval graph

J_7 : $G_1\,G_2\,G_3$
J_5 : $G_1\,G_2\,G_3$
J_6 : $G_2\,G_3$

Figure 2. Gap diagram.

(5) *Consider an empty vector* F*, and set* $i = 1$*, and* $y = x$*;*
(6) *while*$(y \neq s)$
  a) $F(i) = y$*,* $y = P(y)$*,* $i = i + 1$*;*
(7) *Set* $fw_{tmp} = W(x)$ *and* $FV_{tmp} = F$*;*
(8) *Consider the gaps in* F*, construct graph* G *as explained above;*
(9) *Find the maximum independent set* I *in* G*;*
(10) *Add the new jobs corresponding to vertices in* I*; update* fw_{tmp} *and* FV_{tmp} *accordingly;*
(11) *If* $fw_{tmp} \geq fw$*, then set* $fw = fw_{tmp}$ *and* $FV = FV_{tmp}$*;*
(12) *Let* x *be* $J_{\gamma_i+t\alpha}$ *for some* i*, and* t*; delete all vertices* $J_{\gamma_j+t'\alpha}$ *with* $\gamma_j + t'\alpha \geq \gamma_i + t\alpha$ *from* D*, and go to step 2;*
(13) *Output* fw *and the reverse of* FV*.* ◇

4.4.1. Time complexity of shift-based algorithm

The proposed shift-based Algorithm 4.8 enhances Algorithm 4.4 by finding the maximum independent sets in interval graphs. According to Hsiao *et al.*(1992), finding the maximum independent set in an interval graph has the time complexity of less than $O(n^2)$, where n is the number of nodes. Therefore, the shifting process applied at the end of the Algorithm 4.4 does not change its complexity, and the algorithm complexity remains at $O(r^2n^2T/\alpha)$.

4.5. *IPQ-based algorithm*

In this section we propose an algorithm which addresses the problem defined in Section 3.2. Here, we enhance the previous algorithms by considering the pre-specified production targets ($IPQs$) for product families during the scheduling horizon.

In this case, jobs are partitioned into k sets $S_1, S_2, ..., S_k$ according to their families (product types). The scores of all the jobs in each set S_f are the same (i.e. $SS_i = SS_f, \forall J_i \in S_f$). It is desirable to execute $n_f = \lceil IPQ_f \rceil$ jobs of set S_f, for each $1 \leq f \leq k$. Furthermore, we try to maximize the overall schedule score (primary objective), while minimizing the maximum completion time (secondary objective).

The algorithms presented in the previous sections have to be modified to select n_f jobs of set S_f. Therefore, as the score of the maximum path at each step of the new algorithm is updated, we check whether the production target (n_f) has been reached. Let P_u be a path from the start node to vertex u in the network. Note that u corresponds to one of the jobs. Let r_f be the number of jobs from set S_f on path P, and Let $|P|$ denote the number of nodes on path P. The weight of u can then be defined as follows :

$$W(u) = \sum_{i}^{|P|} m_i SS_i$$

where $m_i = 1$ if $r_f \leq n_f$, and $m_i = 0$ when $r_f > n_f$, $\forall J_i \in S_f$. Consequently, when the IPQ_f for some set S_f in path P is reached, the algorithm does not add the weight of vertex v to P, where v corresponds to any job $J_i \in S_f$.

Here is an explanation of the IPQ-based algorithm. Digraph D can be constructed as described in the previous sections. For vertex $u = J_{\gamma_i+t\alpha}$, let $W(u)$ denotes the weight of the maximum path from source s to u and let $P(u)$ denotes the last vertex before u on the maximum path from s to u. At the beginning of the new algorithm, $W(u)$ is set to be the weight of the arc su. Recall that $e(J_{\gamma_i+t\alpha}J_{\gamma_j+t'\alpha})$ is set to be equal to the weight of job J_j. At each step of the algorithm, we consider arbitrary vertex $u = J_{\gamma_i+t\alpha}$ which has not been selected before, and has in-degree zero among all unselected vertices.

Note that there is a vertex u' such that it has been selected in one of the previous steps and $u'u$ is an arc in D. Let P_u be the corresponding maximum path for u, and let r_f be the number of jobs in P_u from product family set of S_f. Set $W(u) = \sum_i^{|P|} m_i SS_i$ where $m_i = 1$ when $r_f \leq n_f$, and $m_i = 0$ otherwise, for $J_i \in S_f$. For every other vertex $v = J_{\gamma_j+t'\alpha}$, replace $W(v)$ by $W(u) + e(uv)$, and set $P(v)$ to be u if the followings hold:

- $W(v) < W(u) + e(uv)$,
- there is no vertex v' with the same associated job as v, on the maximum path from s to u.

Here is the algorithm:

Algorithm 4.9:

(1) $FV = (s)$, $fw = 0$, $Ch = \{s\}$, and $V(D)$ = set of all vertices in D;
(2) For every vertex v, set $W(v) = e(sv)$;
(3) Repeats $(a), (b), (c)$, N times (N is the number of vertices in D):
a) choose vertex $u \notin Ch$, where $W(u)$ is maximum, and u has in-degree zero in $V(D)\backslash Ch$;
b) for every vertex $v \in V(D)\backslash Ch$, where uv is an arc,
c) Suppose v belongs to a job in S_f for some f, Let r_f be the number of jobs from S_f over path P_u; if $r_f < n_f$, then $temp = W(u) + e(uv)$, otherwise $temp = W(u)$; if:
i. $temp > W(v)$,
ii. there is no any other vertex v' on the maximum path from s to u, which both v' and v correspond to the same job,
then set $W(v) = temp$ and $P(v) = u$, update the number of jobs from S_f in P_v. (It needs to copy from u to v);
d) add u to Ch;
(4) Find vertex x where $W(x)$ is maximum;
(5) Consider an empty vector F, and set $i = 1$, and $y = x$;
(6) while $(y \neq s)$,
a) $F(i) = y$, $y = P(y)$, $i = i + 1$;
(7) If $W(x) \geq fw$, $fw = W(x)$ and $FV = F$;
(8) Let x be $J_{\gamma_i + t\alpha}$ for some i, and t; delete all vertices $J_{\gamma_j + t'\alpha}$ with $\gamma_j + t'\alpha \geq \gamma_i + t\alpha$ from D, and go to step 2;
(9) Output fw and the reverse of FV. ◇

4.5.1. Efficiency and effectiveness of IPQ-based algorithm

The proposed IPQ-based Algorithm 4.9 is similar to Algorithm 4.4 on the changes of the score of each node, which is still in the order of $O(rn)$. In addition to save the trace of maximum path P_u from start to vertex u, we also need to have a record of number of vertices that belong to set S_i in P_u. In order to keep these information we use $|D|$ by k array, where k is the number of S_i's. Therefore, the time complexity of IPQ-based algorithm is $O(kr^2n^2T/\alpha)$, where r is the maximum number of basic time grids in time windows intervals, n is the number of jobs, and T is the scheduling time horizon.

Algorithm 4.9 works based on the same approach as Algorithm 4.4. The main difference is in the computing of the weight of the maximum weight path at each step, in which we ignore adding up the score of the jobs if their IPQ target have been met, to the maximum weight path. Therefore, the optimality can be shown similar to Subsection 4.3.2; when all the temporal parameters of the system are multiples of α, then the algorithm finds the optimal solution.

5. Experiments

To assess the performance of proposed scheduling methods, test data sets have been generated to cover a wide range of system structure and characteristics. We used two different structures of networks for each experiment. We then applied the IPQ-based Algorithm and collected the CPU-runtime, and the total schedule score of scheduled

jobs. Here are the two types of the networks:

I. Common Grid Points Network (CGPN): in this network, the time grid points are common for all the jobs as described in Section 4.2. The first possible start time of job i is set to be $\lceil a_i \rceil$, the next grid point time (i.e. $\gamma_i = \lceil a_i \rceil$). There are constant time intervals α between the grids. Consequently, other points will be at $(\lceil a_i \rceil + t\alpha)$. The quality of results may deteriorate by increasing the basic time grid length. Furthermore, based on the structure of the CGPN, if the basic time grid α is very large, the time windows for some jobs may fall within the grid points. In this case, there will be no nodes associated with these jobs in the network, which will result in ignoring these jobs by the algorithm. To improve the quality of the results, we can modify the construction of the network by assigning the grid points to individual jobs.

II. Individual Jobs Grid Points Network (IJGPN): in this network, the grid points are different for each job. Here, the first possible start time of job i is set to be $(\gamma_i = a_i)$. The network are then constructed according to the guidelines explained in Section 4.2. Consequently, other points will be at $a_i + t\alpha$. Since we start the first node for J_i at $(\gamma_i = a_i)$, there will be at least one node for each job in IJGPN network, regardless of the basic time grid length.

We designed our experiments for different numbers of jobs, number of families of jobs, basic grid lengths, schedule scores, time windows, processing times, and setup times. For each scenario, we have produced 5 replicates based on different random seeds.

Data has been generated for two different test categories: effectiveness, and efficiency. The effectiveness test emphasizes more on measuring the quality of the results from each scheduling method by comparing the results with optimal solution. In the efficiency test the emphasis is more on the study of the performance of proposed scheduling methods in terms of CPU run-time. We used C program on a laptop computer with a 2-GHz CPU to run these simulation experiments. During these tests we have carefully observed the robustness of the scheduling methods in terms of their consistency of acceptable performances over a wide range of system's parameters.

The proposed scheduling methods are valid for any scheduling horizon. In this study, typical production shift of 8 hours have been used. Small scheduling horizons result in more frequent re-scheduling of resources, and are recommended for fast-changing environments; or the environments where higher accuracy (smaller time grids) are needed. Long scheduling horizons are recommended for the environments in which the changes are slow and processing times are long. In environments that long scheduling horizon with fine time granularity (small α) is needed, a rolling horizon scheduling system maybe used.

5.1. *Effectiveness test*

In the effectiveness test, we produce scenarios in which optimal solutions are achieved by an exhaustive search (OPT). We then compare the performance of the proposed scheduling methods against the optimal solutions. Due to the complexity of the problem, finding an optimal solution using exhaustive search with large number of jobs are difficult in this case.

Processing times for jobs are generated to be integer numbers with random values up to four different levels (5, 10, 30, 120 mins). Time windows intervals are also integer numbers with random values up to four levels of (10, 30, 120, 240 mins). The experiments are run for different numbers of jobs, number of families of jobs, basic grid lengths,

Table 1. Effectiveness test.

	α=1 (min)			α= 5 (min)			α=15 (min)			α =30 (min)		
Number of Jobs	CGPN-%	IJGPN-%	OPT-RunTime (sec)	CGPN-%	IJGPN-%	OPT-RunTime (sec)	CGPN-%	IJGPN-%	OPT-RunTime (sec)	CGPN-%	IJGPN-%	OPT-RunTime (sec)
2	0.00	0.00	0.00	-1.25	0.00	0.00	-17.50	0.00	0.00	-36.25	0.00	0.00
5	0.00	0.00	0.00	-2.11	-0.42	0.00	-15.50	-0.42	0.00	-26.75	-0.42	0.00
10	0.00	0.00	0.14	-2.24	-0.21	0.14	-12.96	-0.51	0.14	-33.74	-1.58	0.14
12	0.00	0.00	16.40	-1.81	-0.31	17.42	-14.83	-0.41	17.46	-31.79	-1.91	13.41
13	0.00	0.00	183.40	-1.44	-0.08	205.65	-14.05	-1.18	162.11	-28.78	-1.87	171.36

schedule scores, and setup times. For each scenario, five replications are produced based on different random seeds. We collected the total schedule score and the CPU run-time for three methods (CGPN, IJGPN, and OPT). The results of 9600 observations in effectiveness test have been summarized in Table 1. The average run-time for CGPN, and IJGPN for each instance has been less than one second, therefore hasn't been included in this table. The OPT run-time has been reported.

Table 1 shows the quality of schedules produced by CGPN-, and IJGPN-based algorithms compared with the optimal solutions generated by the OPT algorithm. The results are normalized to the optimal solution. Therefore, if for example, the IJGPN-based algorithm was able to produce total score of 48 and OPT score was 50, the IJGPN result is $48 - 50/50 = -4\%$ off the optimal solution.

There are several observations from the results of Table 1: First) in the column where $\alpha = 1$ min, both CGPN- and IJGPN-based algorithms produce optimal solutions. This represents the case where all temporal parameters of the system are multiples of the basic time grid, and was previously addressed in this article; Second) the quality of results from both CGPN and IJGPN-based algorithms deteriorates with the increase of the basic time grid length; Third) IJGPN-based algorithm is less sensitive to the increase of the basic time grid length than CGPN-based algorithm. Even with relatively large α (e.g. 30 min), IJGPN-based algorithm result's average is within 2% of the optimal solution; Fourth) both IJGPN and CGPN-based algorithm average run-times are less than one second, while the OPT method takes relatively long time, and its run-time exponentially increases with the increase of the number of jobs. For n=14 (not reported in the table) it may take several hours for OPT approach for each experiment to produce any results, which shows OPT is not an efficient method.

In general, OPT produces optimal solution, however it is not efficient in terms of run-time in medium or large number of jobs, therefore may not be useful in industrial environments. CGPN produces acceptable results, but is very sensitive to the length of the basic time grid length and its quality deteriorates with the increase of the basic time grid length. Among the tested algorithms, IJGPN shows its superior performance in effectiveness test.

Table 2. Efficiency test, "-" indicates the CPU run-time less than one second.

	α=1 (min)				α=5 (min)				α=15 (min)				α=30 (min)			
Number of Jobs	CGPN-Score	CGPN-RunTime (sec)	IJGPN-Score	IJGPN-RunTime (sec)	CGPN-Score	CGPN-RunTime (sec)	IJGPN-Score	IJGPN-RunTime (sec)	CGPN-Score	CGPN-RunTime (sec)	IJGPN-Score	IJGPN-RunTime (sec)	CGPN-Score	CGPN-RunTime (sec)	IJGPN-Score	IJGPN-RunTime (sec)
5	41.10	-	41.10	-	40.54	-	41.34	-	34.56	-	41.40	-	27.08	-	39.80	-
10	87.96	-	87.96	-	92.31	-	93.08	-	71.59	-	81.76	-	58.68	-	82.63	-
30	213.08	-	213.08	-	212.00	-	213.20	-	156.08	-	182.30	-	132.64	-	178.03	-
50	333.20	-	333.20	-	302.29	-	304.03	-	237.14	-	281.01	-	187.73	-	314.98	-
100	524.49	2.94	524.49	2.99	491.34	-	508.41	-	369.54	-	490.10	-	236.83	-	474.34	-

5.2. *Efficiency test*

The focus of this test is to study the efficiency of proposed approaches in-terms of the CPU run-time required to generate schedules. Since the proposed algorithms are designed to run at the beginning of short scheduling horizons, it is important that the schedules can be efficiently produced. In this test, the system parameters range are similar to those discussed in the effectiveness test. Similar to the previous test, each scenario has been replicated 5 times with different random seeds. However, here the number of jobs are many more than the previous test (i.e. more than 100 jobs in some cases). As discussed before, OPT takes very long time in large number of jobs and is not practical for our purposes, therefore OPT results hasn't been reported here. The efficiency test results are summarized in Table 2.

The results show the changes of CPU run-time verses the changes in the number of jobs, and the basic time grid. As shown, the decrease in the value of the basic time grid length will result in longer run-time. The increase in the number of jobs also will results in the longer run-times.

The efficiency study shows that even with the large number of jobs and fine time grids, both CGPN and IJGPN- based algorithms are efficient in producing schedules in acceptable run-time. CGPN-based algorithm slightly outperforms IJGPN-based algorithm in terms of run-time, because it may have less number of nodes. Similar to the effectiveness test, however, IJGPN produces better quality results in large basic time grids.

In conclusion, based on our study and the efficiency and effectiveness tests, OPT is a very slow scheduling approach and therefore not implementable in most systems. Both IJGPN and CGPN-based approaches are efficient and therefore suitable for our purposes. CGPN-based approach is slightly more efficient than IJGPN-based approach, but the effectiveness of CGPN-based approach in terms of producing quality results deteriorates rapidly with the increase of the basic time grid. Therefore, IJGPN-based algorithm shows its superior performance in-terms of producing quality schedules in acceptable time.

6. Summary and closing remarks

In this paper, we demonstrated how to develop network-based scheduling algorithms to sustain the desired WIP profile in manufacturing environments. We used short-term production targets to coordinate decentralized local schedulers and to make the objectives

of specific areas inline with the chain objectives.

We introduced a basic network-based algorithm to select and schedule several jobs on a single machine with sequence dependent setup times and strictly enforced time windows constraints on the start-times of each job. We studied some of the properties of the solution in respect to changes in system parameters. We introduced a shift-based algorithm which enhances the basic algorithm by shifting some of the jobs and inserting more jobs to improve the solution quality. Finally, we included non-homogenous family of jobs, and used Ideal Production Targets in the IPQ-based algorithm.

We modeled wide range of test problems with two different network structures; namely, Common Grid Points Network (CGPN) and Individual Job Grid Points Network (IJGPN). In CGPN, all the jobs use the time grid points that are common for all of them. In IJGPN, each job has its own time grid points. We compared the effectiveness and the efficiency of the proposed algorithms in terms of the quality of results and the running time to generate schedules. As a benchmark, we also included the results of a simple exhaustive search (OPT) algorithm in our analysis.

Our study shows the proposed algorithms substantially outperform the OPT algorithm in terms of the efficiency of producing results in terms of the required time for generating any schedules. Both CGPN and IJGPN-based algorithms produce optimal results when the temporal parameters of the system are integer multiples of the basic time grid. These algorithms produce results that are close to optimal solution where the basic time grid is small under a broad variety of scenarios. CGPN-based algorithm shows a slight advantage in the efficiency test over IJGPN-based algorithm, but its quality of results deteriorate rapidly with the increase of the basic time grid length. Therefore, among the tested methods, IJGPN-based algorithm appears to be a reliable method with acceptable levels of efficiency and effectiveness.

Several directions for future research are apparent from this study. Firstly, in this study the basic time grid is considered to be the same and constant for all the jobs. Studies can be conducted on situations in which dynamic time grids are used for jobs. In these cases, the basic time grid can be adjusted for each job according to the scheduling needs. Secondly, this research could be extended to address both homogenous and non-homogenous multiple machine environments. Thirdly, we developed our approaches based on network flow methods. Further studies could be done on using mathematical programming methods (e.g. Mixed Integer Programming), and developing efficient heuristics to generate schedulers. Finally, similar approaches could be developed to address the scheduling challenges in service industries. The efficiency and effectiveness of these approaches should be further studied under different scenarios.

References

[1] Ahmadi, R.H., and Bagchi, U., 1992. Minimizing job idleness in deadline constrained environments. *Operations Research*, 40(5), 972-985.

[2] Allahverdi, and A., Gupta, JND., Aldowaisan, T., 1999. A review of scheduling research involving setup considerations. *OMEGA- International Journal of Management Science*, 27(2), 219-239.

[3] Allahverdi, A., Ng, C.T., Cheng, TCE., and Kovalyov, MY., 2008. A survey of scheduling problems with setup times or costs. *European Journal of Operationl Research*, 187 (3), 985-1032.

[4] Asano, M., and Ohta, H., 1996. Single machine scheduling using dominance relation

to minimize earliness subject to ready and due times. *International Journal of Production Economics*, 44 (1-2), 35-43.

[5] Ascheuer, N., and Fischetti, M., Grötschel M., 2001. Solving the asymmetric traveling salesman problem with time windows by branch-and-cut. *Mathematical Programming Series A*, 90(3), 475-506.

[6] Belouadah, H., Posner, ME., and Potts, CN., 1992. Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete Applied Mathematics*, 36(3), 213-231.

[7] Bianco, L., and Ricciardelli, S., 1982. Scheduling of a single machine to minimize total weighted completion time subject to release times. *Naval Research Logistics Quarterly*, 29(1), 151-167.

[8] Carlton, WB., and Barnes, JW., 1996. Solving the traveling-salesman problem with time windows using tabu search. *IIE Transactions*, 28(8), 617-629.

[9] Chand, S., and Schneeberger, H., 1988. Single machine scheduling to minimize weighted earliness subject to no tardy jobs. *European Journal of Operational Research*, 34(2), 221-230.

[10] Dumas, Y., Desrosiers, J., Gelinas, E., and Solomon, MM., 1995. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43(2), 367-371.

[11] Garey, MR., and Johnson, DS., 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA.

[12] Gendreau, M., Hertz, A., Laporte, G., and Stan, M., 1998. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3), 330-335.

[13] Gélinas, S., and Soumis, F., 1997. A dynamic programming algorithm for single machine scheduling with ready times. *Annals of Operations Research*, 69(10), 135-156.

[14] Golumbic, M., 1980. *Algorithmic, Graph Theory and Perfect Graphs.* Academic Press, New York.

[15] Hariri, AMA., and Potts, CN., 1983. Algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Applied Mathematics*, 5(1), 99-109.

[16] Hsiao, J.Y., Tang, C.Y., and Chang, R.S., 1992. An efficient algorithm for finding a maximum weight 2-independent set on Interval Graphs. *Information Processing Letters*, 43(5), 229-235.

[17] Jula, P., and Leachman, R.C., 2008. Coordinating decentralized local schedulers in complex supply chain manufacturing. *Annals of Operations Research*, 161(1), 123-147.

[18] Jula, P., and Leachman, R.C., 2010. Coordinated multi stage scheduling of parallel batch processing machines under multi resource constraints. *Operations Research*, 158(4), 933-947.

[19] Kedad-Sidhoum, S., and Sourd, F., 2010. Fast neighborhood search for the single machine earliness-tardiness scheduling problem. *Computers & Operations Research*, 37(8), 1464-1471.

[20] Leachman, R.C., Kang, J., and Lin.,V., 2002. SLIM: short cycle time and low inventory in manufacturing at samsung electronics. *Interfaces*, 32(1), 61-77.

[21] Lenstra, J.K., Rinnooy Kan, A.H.G., and Brucker, P., 1977. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1, 343-362.

[22] Li, X.P., Wang, Q., and Wu, C., 2008. Heuristic for no-wait flow shops with

makespan minimization. *International Journal of Production Research*, 46(9), 2519-2530.

[23] Pan, Y., and Shi, L., 2005. Dual constrained single machine sequencing to minimize total weighted completion time. *IEEE Transactions on Automation Science and Engineering*, 2(4), 344-357.

[24] Posner, M., 1985. Minimizing weighted completion times with deadlines. *Operations Research*, 33(3), 562-574.

[25] Potts, CN., and Wassenhove, LN., 1983. Algorithm for single machine sequencing with deadlines to minimize total weighted completion time. *European Journal of Operational Research*, 12(4), 379-387.

[26] Ruiz, R., and Allahverdi, A., 2009. New heuristics for no-wait flow shops with a linear combination of makespan and maximum lateness. *International Journal of Production Research*, 47(20), 5717-5738.

[27] Smith, WE., 1956. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2), 59-66.

[28] Sourd, F., 2006. Dynasearch for the earliness-tardiness scheduling problem with release dates and setup constraints. *Operations Research Letters*, 34 (5), 591-598.

[29] Zhu, XY., and Wilhelm, WE., 2006. Scheduling and lot sizing with sequence-dependent setup: A literature review Source. *IIE Transactions*, 38(11), 987-1007.