A generalized resource constrained production scheduling problem: An evolutionary algorithm approach

Olli Kämäräinen [1] *Vesa Ek* [2] *Kimmo Nieminen* [1] *Sampo Ruuth* [1]

[1] *Systems Analysis Laboratory, Helsinki University of Technology, P.O.Box 1100, 02015 HUT, Finland*

[2] *Tieto Corporation, Espoo, Finland*

April 1999

Abstract

Many real world scheduling problems are large combinatorial optimization problems and can not generally be solved optimally in a reasonable computing time. We develop a genetic algorithm implementation for generalized resource constrained scheduling problems, where the objective is to schedule the operations subject to ready times, due dates, precedence relations and resource constraints in order to optimize the objective function. The scheduling procedure has two levels. A genetic algorithm is first used for determining the priorities of the operations. Then the schedule is calculated based on the priorities. The implementation will be applied to production scheduling problem of a silicon wafer manufacturer. Some experimental results with our algorithm will be discussed.

Key words: Genetic algorithm, Production scheduling, Combinatorial optimization

1 Introduction

Production scheduling is an important part of the production control of a manufacturing company. By production scheduling we find the right sequence to do the jobs and the right schedule, when each operation of the job should be processed at each stage of the production process.

Traditional project scheduling methods, such as PERT and CPM, are not enough for production scheduling, because they produce infinite schedules, ie they can not take resource constraints into account, and this may lead to non-feasible solutions. A schedule is called feasible, if the precedence relations of the operations are maintained and the resource and other constraints are met. More efficient methods to solve resource constrained scheduling problems beforehand are needed.

In practice, production scheduling often relies on priority based sequencing to determine which job a machine should process at the time, when the machine becomes available. These dispatching algorithms are simple and flexible, but they lack optimality because they make current decisions without considering future events and what is happening at other machines. This can cause long lead times and big stocks in highly utilized factories.

Many real world scheduling problems are large scale combinatorial optimization problems and can not generally be solved optimally in a reasonable computing time. Therefore, heuristics based search methods, such as tabu search, simulated annealing and genetic algorithms, have been used to give solutions.

This paper deals with a scheduling procedure for generalized resource constrained scheduling problem, where objective is to schedule the operations subject to ready times, due dates, precedence relations and resource constraints in order to optimize the objective function. The procedure is based on search heuristics from Lee and Kim [3] for resource constrained project scheduling problem. The algorithm has two levels. First, a genetic algorithm determines the priorities of the operations. Then the priorities are used for calculating the schedule.

The scheduling procedure will be applied to production scheduling problem of a Finnish silicon wafer manufacturer. We are developing a scheduling system, which will be integrated with the production control system of the company.

The paper is organized as follows. In Chapter 2 the generalized resource constrained scheduling problem is introduced. Chapter 3 describes the scheduling procedure with genetic algorithm, Chapter 4 provides some computational results and Chapter 5 includes the conclusions.

2 Generalized resource constrained scheduling problem

Production scheduling problem can be considered as a project, which has to be scheduled. We have n jobs J_i, $i = 1, \ldots, n$, each of which consists of $n(i)$ operations $O_{i1}, \ldots, O_{in(i)}$. We need also dummy operations O_0 and O_*. The operation O_0 is the first operation in the project and its successors are the first operations of every job. O_* is the project's last operation with the last operations of all jobs in its predecessors. A network example is illustrated in Figure 1, where jobs J_1, J_2 and J_3 consist of operations $O_{11}, \ldots, O_{19}$, $O_{21}, \ldots, O_{24}$ and $O_{31}, \ldots, O_{37}$.

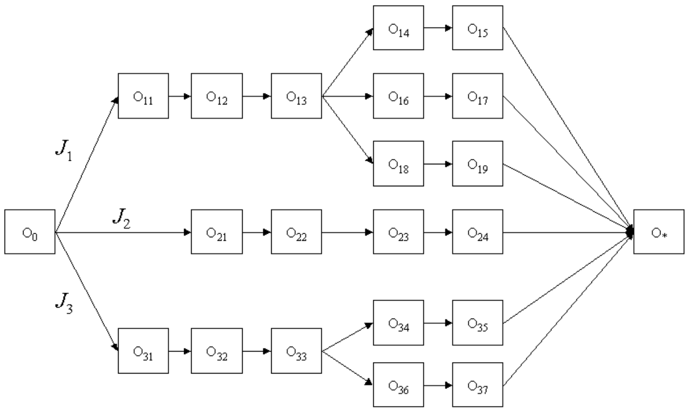

Figure 1: Network of operations

The start time of operation O_{iq} is denoted as s_{iq} and the finish time is f_{iq}. The duration is correspondingly $d_{iq} = f_{iq} - s_{iq}$. The durations d_0 and d_* of the dummy operations O_0 and O_* are zero, and thus their start and finish times are equal: $s_0 = f_0$ and $s_* = f_*$. The problem is to solve the operation finish times $\mathbf{f} = (f_{11}, \ldots, f_{1n(1)}, f_{21}, \ldots, f_{2n(2)}, \ldots, f_{n1}, \ldots, f_{nn(n)})$ in order to optimize the objective function $J(\mathbf{f})$. Because the finish time of an operation equals to the known duration of an operation added to the start time, ie $f_{iq} = s_{iq} + d_{iq}$ for all operations O_{iq}, the problem can be considered as solving the start times $\mathbf{s} = (s_{11}, \dots, s_{1n(1)}, s_{21}, \dots, s_{2n(2)}, \dots, s_{n1}, \dots, s_{nn(n)})$ in order to optimize the objective function $J(\mathbf{s})$ as well.

The time scale is discretized into constant intervals $\Delta t = t_i - t_{i-1}$ starting from the time t_0. For convenience, the time unit Δt is scaled to 1 and the start time t_0 is adjusted to zero. We have K different resource types $M_k, k = 1, \dots, K$. Operation O_{iq} requires $r_{iq,k}$ number of renewable resource type M_k during the execution. The resource availability depends on time. It is denoted by $a_{k,t}$, and equals to the available number of resource type M_k during time period $(t-1, t]$.

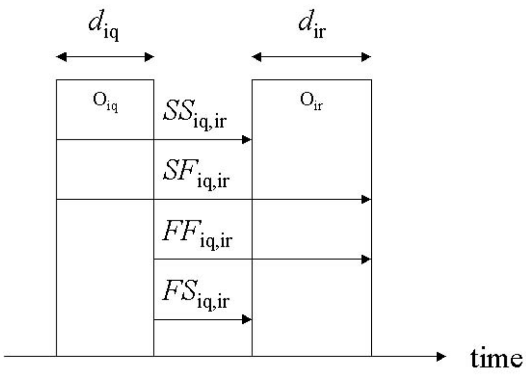

Figure 2: Precedence relations

There are four possible precedence relations between predecessors and successors. The finish-to-start constraint $FS_{iq,ir}$ is the most typical precedence relation and it refers to the minimum number of time units after the completion f_{iq} of the predecessor O_{iq} and before the start s_{ir} of successor O_{ir}. The start-to-start constraint $SS_{iq,ir}$ denotes the minimum number of time units that must be completed on O_{iq} before the successor O_{ir} is started. The start-to-finish constraint $SF_{iq,ir}$ is equal to the minimum number of time units between the start of O_{iq} and the finish of O_{ir}. The finish-to-finish constraint $FF_{iq,ir}$ tells the minimum number of time units that have to be completed on the successor O_{ir} after the completion of the predecessor O_{iq}. In practice, start-to-finish and finish-to-finish relations are needed very seldom in production scheduling. All of these four time lags are fixed beforehand. The precedence constraints can be formulated as follows:

$$s_{iq} + SS_{iq,ir} \leq s_{ir}, \forall (O_{iq}, O_{ir}) \in H_1 \tag{1}$$
$$s_{iq} + SF_{iq,ir} \leq f_{ir}, \forall (O_{iq}, O_{ir}) \in H_2 \tag{2}$$
$$f_{iq} + FS_{iq,ir} \leq s_{ir}, \forall (O_{iq}, O_{ir}) \in H_3 \tag{3}$$
$$f_{iq} + FF_{iq,ir} \leq f_{ir}, \forall (O_{iq}, O_{ir}) \in H_4 \tag{4}$$

H_1 denotes the set of pairs of operations indicating start-to-start relations, and correspondingly, H_2, H_3 and H_4 are sets of operation pairs indicating start-to-finish, finish-to-start and finish-to-finish relations. Because only one of the precedence constraints (1), (2), (3) and (4) between two operations is significant at the time, they can be converted into a finish-to-start relation using the conversion formula (5). The logic of the conversion can be easily understood with help of Figure 2.

$$FS'_{iq,ir} = \max\{SS_{iq,ir} - d_{iq}, SF_{iq,ir} - d_{iq} - d_{ir}, FS_{iq,ir}, FF_{iq,ir} - d_{ir}\} \tag{5}$$

Generalized resource constrained scheduling problem includes also ready time and due date constraints for an individual operation (see (9) and (10)). The ready time g_{iq} is the earliest possible time, when operation O_{iq} can be started. Respectively, the due date h_{iq} is the latest possible time to finish the operation. The whole generalized resource constrained scheduling problem is formulated as follows (based on [1]):

$$\begin{array}{rcll}
\text{Optimize} & J(\mathbf{f}) & & (6)\\
\text{subject to} & & & \\
s_0 & = & 0 & (7)\\
f_{iq} + FS'_{iq,ir} & \leq & s_{ir}, \forall (O_{iq}, O_{ir}) \in H & (8)\\
s_{iq} & \geq & g_{iq}, \forall O_{iq} & (9)\\
f_{iq} & \leq & h_{iq}, \forall O_{iq} & (10)\\
\sum_{O_{iq} \in S_t} r_{iq,k} & \leq & a_{k,t}, M_k = 1, \ldots, K \text{ and } t = 1, \ldots, f_* & (11)
\end{array}$$

where

$$\begin{array}{rcl}
s_{iq} & = & \text{start time of operation } O_{iq}\\
f_{iq} & = & \text{finish time of operation } O_{iq}\\
FS'_{iq,ir} & = & \text{minimum time lag between operations } O_{iq} \text{ and } O_{ir}\\
 & & \text{indicating precedence constraints (see (5))}\\
H & = & \text{set of all pairs of operations indicating precedence constraints}\\
g_{iq} & = & \text{ready time of operation } O_{iq}\\
h_{iq} & = & \text{due date of operation } O_{iq}\\
S_t & = & \text{set of operations in progress in time interval } (t-1, t]\\
 & = & \{O_{iq} | s_{iq} < t \leq f_{iq}\}\\
r_{iq,k} & = & \text{amount of resource type } M_k \text{ required by operation } O_{iq}\\
a_{k,t} & = & \text{availability of resource type } M_k \text{ during time interval } (t-1, t]\\
K & = & \text{number of resource types}
\end{array}$$

In our application for silicon wafer production scheduling, the durations and the resource requirements of the operations are based on batch sizes and product specifications, and are computed automatically before scheduling. Because the durations are known, possible setup times can be added using appropriate precedence relations. Although the production scheduling problem in our partner company can be formulated into generalized resource constrained problem, this is not always the case in general. Sometimes an operation may be interrupted and resumed at a later time. This *operation pre-emption* may result eg in a shorter makespan, when resource availability is limited. In *multi-mode* resource constrained scheduling, operation durations depend on committed resources. Furthermore, the resource requirements of operation may be variable over time horizon.

3 Scheduling procedure

The scheduling algorithm is based on the procedure presented by Lee and Kim [3] for solving resource constrained project scheduling problem (ie generalized resource constrained project scheduling problem, where all precedence constraints are finish-to-start type with zero minimum lag, resource availabilities are constant over time and the ready time and the due date constraints (9) and (10) are not taken into account). The algorithm consists of two phases. First, the priorities of the operations are determined and after that the schedule is calculated based on the priorities of the operations.

3.1 Genetic algorithm

The genetic algorithm is used for determining the priorities of the operations. The array of integers, which is called a *chromosome*, consists of the priority of each operation. The integer denoting the priority is called a *gene*. A set of chromosomes is called a *population* and a population at a given time is called a *generation*. The evaluation function, which can be considered as an objective function of the problem, measures the quality of the solution given by a chromosome. The goal of scheduling is to construct a feasible schedule, which optimizes the chosen evaluation function. We use a simple version of genetic algorithm, and it can be formulated as follows:

1. Initialize the first generation, ie give a random integer to each gene of each chromosome of the population so that two genes in the same chromosome can not have the same value.
2. Create the next generation:
   - Initialize the new generation by duplicating the previous generation.
   - Find the best chromosome, ie the chromosome with the best evaluation value, of the previous generation. The evaluation function for each chromosome is calculated by the priority scheduling procedure.
   - Divide the population into pairs of chromosomes. Let l be the index of a pair. Repeat the next steps for all pairs l.
     - Select randomly a chromosome from the previous generation and a crossover point. The crossover point lies between two consecutive genes.
     - Crossover: Swap the values of genes (ie priorities of operations) after the crossover point between the randomly selected chromosome and the best chromosome with a probability called crossover rate. Replace the pair l with the offspring chromosomes.
     - Mutation: Swap two randomly selected genes of a chromosome with a probability called mutation rate. Apply the mutation to the both chromosomes in the pair l.
   - Replace the last chromosome of the population with the best chromosome of the previous generation. This is called an *elitism* and it guarantees that the next generation will contain a chromosome, which is better than or equal to the best chromosome in the previous generation.
3. If the termination condition is met, then stop. Otherwise, go to step 2.

3.2 Priority scheduling procedure

The priority scheduling procedure is given below. It describes how to construct the schedule based on the priorities given by a chromosome. If a population contains N chromosomes, there are correspondingly N schedules. The procedure below is for forward scheduling, but it can be easily converted into backward scheduling.

1. Determine the priorities of operations using the genetic algorithm. Initialize the current time t_{now} to the schedule start time t_0. Initialize the *resource availability vector* $\mathbf{R}$, whose component R_k denotes the availability of resource type M_k at the current time t_{now}. If no operations are scheduled at the time t_{now}, R_k equals $a_{k,t_{now}}$ ie the number of available resource type M_k in time period $(t_{now} - 1, t_{now}]$ before scheduling.

2. If all the operations are scheduled, then stop. Otherwise, create the set E, which consists of the non-scheduled operations,

   - whose predecessors have already been scheduled so that t_{now} is more than the minimum lag FS' added to their finish times,
   - which do not need any resource type M_k more than R_k at the moment t_{now} and the required resources are available over the operating period start from the moment t_{now} and
   - whose ready times are not more than t_{now}.

   If the set E is empty, go to step 3. Otherwise, go to step 4.

3. Adjust the t_{now} to the next time, when resource availability conditions or precedence relations will change. This is the earliest time of the following alternatives:

   - The earliest of the finish times of the operations currently being processed
   - The earliest time, when availability of some resource type increases
   - The next possible start time of operations, whose predecessors have already been scheduled, considering minimum lags and ready times

   Update the vector $\mathbf{R}$ to correspond the resource availabilities at the time t_{now}. Go to step 2.

4. Select the operation with the highest priority from the set E. Set the start time of the operation to be t_{now}. Update the resource availability vector $\mathbf{R}$, ie do the reductions to component R_k, if the scheduled operation requires resource type M_k. Go to step 2.

In Kolisch and Hartmann [2], this kind of time incrementation procedure is classified as a *parallel schedule generation scheme.* The priority scheduling procedure above produces a feasible solution with precedence and resource constraints (8) and (11), if it exists. Depending on the scheduling direction, either ready time or due date constraints can be taken into account in the priority scheduling phase. A forward schedule is feasible with the ready time constraints, but it may be non-feasible with the due dates. Correspondingly, despite of managing the due dates, a backward schedule may have problems with the ready time constraints.

In forward scheduling, the due date constraints for individual operations (10) are taken into account by using a penalty term in the objective function $J(\mathbf{f})$. For example, if we are minimizing the makespan of the project, the objective function is presented as follows:

$$\text{Minimize } J(\mathbf{f}) = f_* + \sum_{\forall J_i} w_i \sum_{q=1}^{n(i)} \max\{0, f_{iq} - h_{iq}\} \tag{12}$$

The makespan is optimized by minimizing the finish time of the last dummy operation O_*. The tardiness $\max\{0, f_{iq} - h_{iq}\}$ of operation $O_{iq} \in \{O_{i1}, \ldots, O_{in(i)}\}$ is multiplied by the penalty weight of job J_i. Correspondingly, in backward scheduling, the penalty terms are used for coping with the ready time constraints. In the case of makespan minimization, the start time of the dummy operation O_0 is maximized and each earliness $\max\{0, g_{iq} - s_{iq}\}$ is penalized by weight w_i of job J_i. The backward scheduling objective function is formulated as follows:

$$\text{Maximize } J(\mathbf{f}) = s_0 - \sum_{\forall J_i} w_i \sum_{q=1}^{n(i)} \max\{0, g_{iq} - s_{iq}\} \tag{13}$$

4 Computational experiments

4.1 Test data

We have made preliminary experiments with the scheduling procedure in order to become familiar how large problems could be solved in a reasonable computing time. The experiments have been run with the following test cases: First, the number of jobs n is selected. Every job has 14 consecutive operations, whose durations are generated from a discrete uniform distribution with range $[1, 8]$. The minimum lags between consecutive operations are selected randomly from a discrete uniform distribution with range $[-1, 5]$. We have also 14 resource types and each operation requires one resource unit so that eg the first operation of a job uses the first resource type, the second operation needs the second resource type etc. The availability of a resource type varies between 0 and $2n/100 + 1$. The time intervals between the availability changes vary between 1 and 100 (all of them are generated from a discrete uniform distribution).

The ready times and the due dates are generated as follows: At first, we select for each job $J_i, i = 1, \ldots, n$ a random integer b from an uniform distribution with range $[1, 10n]$. The due date for the last operation O_{i14} of the job equals b added to the constant 720 and the smallest possible duration of the job. For every fifth job, we select randomly also another operation O_{iq} between O_{i1} and O_{i13} and set a due date, which is b added to the constant 720 and the minimum duration from O_{i1} to O_{iq}. For every seventh job, we select again $O_{iq}, q = 1, \ldots, 13$ from a discrete uniform distribution and set a ready time to b added to the minimum duration from O_{i1} to O_{iq-1} (without adding 720).

The scheduling direction is forwards, and the objective is to minimize the makespan. In the objective function (12), all the penalty weights w_i are set to 20. The crossover and the mutation rate are both set to 0.5. The procedure terminates, when the computing time exceeds 30 minutes. The tests are run for problem sizes 2800, 4200, 5600, 7000 and 8400 with population sizes 100 and 200. The procedure is implemented with C++ language and the test runs are executed on a personal computer with two 450 MHz Pentium II processors.

4.2 Results

The computational results are shown in Table 1. The generation, in which a feasible solution with the due dates is found, is presented for each combination of the problem size and the population size. Also, the average tardiness per job from the first and the final generation are provided. The results indicate, that the algorithm decreases tardiness, and a feasible solution seems to be found soon. In most cases, the makespan does not improve after the initial generation. However, the convergence of the algorithm is highly dependent on the test cases and the algorithm parameters, including the termination condition. Table 2 provides approximate computational time of calculating the evaluation function for one chromosome.

Table 1:

Problem size	2800		4200		5600		7000		8400	
Population size	100	200	100	200	100	200	100	200	100	200
Number of generations	81	40	40	20	23	11	15	7	9	4
Feasible solution found	25	12	-	-	14	-	15	2	-	-
Av. tard. per job (first)	1.00	0.91	2.85	2.32	0.48	0.23	0.02	0.08	3.28	2.92
Av. tard. per job (final)	0	0	1.68	1.11	0	0.02	0	0	2.80	2.25

Table 2:

Problem size	2800	4200	5600	7000	8400
Comp. time for one chromosome (sec)	0.22	0.44	0.76	1.15	1.81

5 Summary

In this paper, we proposed a procedure for solving a generalized resource constrained scheduling problem. In the procedure, the schedule is calculated based on priorities of operations. Priorities are generated using a genetic algorithm.

The results of preliminary computational experiments are promising and our scheduling procedure could be applied to relatively large problems, which is the case in real life production scheduling. A reasonable computing time is important so that frequent rescheduling is possible in the dynamic production environment with external and internal disturbances. However, further testing is needed in order to improve the genetic algorithm and to select the right parameters.

References

[1] E.L. Demeulemeester, W.S. Herroelen, A Branch-and-bound procedure for the generalized resource-constrained project scheduling problem, Operations Research 45 (2) (1997) 201-212.

[2] R. Kolisch, S. Hartmann, Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis, in: J. Weglarz (Ed.), Handbook on recent advances in project scheduling, Kluwer, Amsterdam, to appear.

[3] J.-K. Lee, Y.-D. Kim, Search Heuristics for Resource Constrained Project Scheduling, Journal of Operations Research 47 (5) (1996) 678-689.