Large scale generalized resource constrained scheduling problems: A genetic algorithm approach

Olli Kämäräinen [1]

Vesa Ek [2]

Kimmo Nieminen [3]

Sampo Ruuth [3]

[1] IC-Parc, Imperial College, London, UK

[2] TietoEnator Corporation, Espoo, Finland

[3] Systems Analysis Laboratory, Helsinki University of Technology, Finland

Abstract

Many scheduling problems in production environment are large scale combinatorial optimization problems and cannot generally be solved optimally in reasonable computing time. We describe a genetic algorithm procedure for generalized resource constrained scheduling problems, where the objective is to schedule the operations subject to ready times, due dates, precedence relations and time-dependent resource constraints in order to optimize the given objective function. The scheduling procedure has two levels. A genetic algorithm is first used to determining the priorities of the operations. Then the schedule is calculated based on the priorities. Some experimental results will be discussed.

AMS Subject Classification: Primary 90B35 (Scheduling theory, deterministic),

Secondary 68T20 (Problem solving (heuristics, search strategies, etc.))

Key words: Production scheduling, Genetic algorithms, Combinatorial optimization

1. Introduction

Production scheduling is an important part of the production planning of many manufacturing companies. By scheduling it is possible to 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 scheduling methods, such as PERT and CPM, are not enough for production scheduling, because they produce infinite schedules, i.e. they can not take resource constraints into account. Infinite scheduling may give results, which are not feasible. A schedule is called feasible, if the precedence relations of the operations are maintained and the resource and other constraints are satisfied. Resource constrained production scheduling is called *finite capacity scheduling*. In manufacturing industry, efficient methods to solve resource constrained scheduling problems are needed.

In practice, production scheduling often relies on priority based sequencing in order 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 usually make current decisions without considering future events and what is happening at other machines. This can cause long lead times and big inventories in highly utilized manufacturing facilities [1].

However, many scheduling problems in real production environment are large scale combinatorial optimization problems and can not generally be solved optimally in a reasonable computing time. Therefore, metaheuristic search methods, such as tabu search, simulated annealing and genetic algorithms, have been used to finding solutions.

This paper deals with a genetic algorithm based scheduling procedure for generalized resource constrained scheduling problem, where the objective is to schedule the operations subject to ready times, due dates, precedence relations and time-dependent resource constraints in order to optimize the objective function. 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 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 summary.

2. Generalized resource constrained scheduling problem

There are different types of project scheduling problems, and in many cases, they are suitable for production scheduling. For example, the generalized resource constrained project scheduling problem (GRCPSP) described in Demeulemeester and Herroelen [2] is a convenient formulation for this purpose. We have n jobs J_i , $i = 1, \dots ,$ n, each of which consists of n_i operations O_{i1} , … , O_{in_i}. The processing order of operations is determined in advance. Dummy operations O_0 and O_* link the jobs together to be one project. An example of production network is illustrated in Figure 1, where jobs J_1, J_2 and J_3 consist of operations O_{11} ,…,O_{19}, O_{21} ,…, O_{24} and O_{31} ,…, O_{37}.

Figure 1.

The start time of operation O_{iq} is denoted as s_{iq} and the finish time is f_{iq}. The duration is known beforehand and it is $d_{iq} = f_{iq} - s_{iq}$. 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 objective is to solve finish times $\mathrm{f} = (f_{11} , \dots , f_{1n_1} , f_{21} , \dots , f_{2n_2} , \dots , f_{n1} , \dots , f_{nn_n})$ of operations O_{i1} , … , O_{in_i} for each job J_i , $i = 1, \dots , n$ in order to optimize the objective function $J(\mathrm{f})$. Because the finish time of an operation equals to the known duration of an operation added to the start time, i.e.

$f_{iq} = s_{iq} + d_{iq}$ for all operations O_{iq}, the problem can be considered as solving start times s = (s_{11} , … , s_{1n_1} , s_{21} , … , s_{2n_2} , … , s_{n1} , … , s_{nn_n}) in order to optimize the objective function J(s) as well.

There are four possible precedence relations between predecessor O_{iq} and successor O_{ir}, finish-to-start $FS_{iq,ir}$, start-to-start $SS_{iq,ir}$, finish-to-finish $FF_{iq,ir}$ and start-to-finish $SF_{iq,ir}$. They tell desired minimum lags between successors and predecessors, e.g. in the case of a start-to-start relation, successor O_{ir} should not be started before a lag sized $SS_{iq,ir}$ added to start time s_{iq} of predecessor O_{iq}. All of these four time lags are fixed beforehand. Because the duration d_{iq} of each operation O_{iq} is known beforehand, all the lags from precedence relations can be combined using the following formula [2].

$$FS'_{iq,ir} = \max\{SS_{iq,ir} - d_{iq}; SF_{iq,ir} - d_{iq} - d_{ir}; FS_{iq,ir}; FF_{iq,ir} - d_{ir}\} \tag{1}$$

The generalized resource constrained scheduling problem is now formulated as follows:

$$\text{Optimize} \qquad J(\mathrm{f}) \tag{2}$$

$$\text{subject to} \qquad \begin{cases} s_0 = t_0 \text{ , in forward scheduling} \\ f_* = t_E \text{ , in backward scheduling} \end{cases} \tag{3}$$

$$s_{ir} \geq f_{iq} + FS'_{iq,ir} , \qquad \forall (O_{iq}, O_{ir}) \in H, \tag{4}$$

$$s_{iq} \geq g_{iq} , \qquad \forall O_{iq} \tag{5}$$

$$f_{iq} \leq h_{iq} , \qquad \forall O_{iq} \tag{6}$$

$$\sum_{O_{iq} \in S_t} r_{iq,k} \leq a_{k,t} , \qquad t = 1,2,\ldots,f_*, k = 1,2,\ldots,K \tag{7}$$

The scheduling direction can be forward or backwards. In forward scheduling, start time s_0 of the first dummy operation is fixed to t_0. Correspondingly, in backward scheduling, finish time f_* of the last dummy operation is fixed to t_E. Precedence constraints (4) set start time s_{ir} of successor O_{ir} to be more than finish time f_{iq} of predecessor O_{iq} and converted minimum lag $FS'_{iq,ir}$ between operations O_{iq} and O_{ir}. H denotes a set of all pairs of operations O_{iq} and O_{ir} indicating precedence constraints. According to ready time and due date constraints (5) and (6), operation O_{iq} may not be started before ready time g_{iq} and finished after due date h_{iq}. 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 renewable resource types (such as machines) M_k, $k = 1, \dots, K$ and operation O_{iq} requires resource type M_k a fixed number of $r_{iq,k}$.. In (7), the availability $a_{k,t}$ of resource type M_k in time interval $(t\text{-}1, t]$ may not be exceeded. S_t denotes the set of operations in progress during time interval $(t\text{-}1, t]$. In this formulation, operations may not be interrupted during the execution and resumed at a later time.

3. Scheduling procedure

The algorithm consists of two phases. First, the priorities of the operations are determined. After that, the schedule is calculated based on the priorities of the operations.

3.1. Genetic algorithm

A genetic algorithm is used in determining the priorities of the operations. A vector of integers, which is called a *chromosome*, consists of the priorities of the operations to be scheduled. In evolutionary computing terms, an integer denoting a decision variable is called a *gene*, which now represents a priority of an operation. This representation is used e.g. in the scheduling procedure described in Lee and Kim [4]. A set of chromosomes is called a *population* and a population at a given time is called a *generation*. For example, if the population consists of N chromosomes and a chromosome consists of M genes, then there is N integer vectors with size M.

An *evaluation function* measures the quality of the solution given by a chromosome. The purpose of an evaluation function is to assign a numerical value to a chromosome. The evaluation function can be considered as an objective function of the optimization problem. Here the evaluation function for each chromosome is calculated by the priority scheduling procedure formulated in Chapter 3.2. 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. The initial generation is created, i.e. a random integer is given to each gene of each chromosome of the population so that two genes in the same chromosome does not have the same value.
2. Create the next generation:
   - Initialize the new generation by duplicating the previous generation.
   - Find the best chromosome, i.e. the chromosome with the best evaluation value, of the previous generation.
   - 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 (i.e. the priorities of the operations) after the crossover point between the randomly selected chromosome and the best chromosome with a probability called a crossover rate. Replace the pair *l* with the two new 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.

There are several other ways to use a genetic algorithm. E.g. the initial population could be created by some effective priority rules. In reproduction, a random chromosome from previous generation could be chosen and taken to the new population e.g. with a probability given by the evaluation function [4]. In crossover, instead of only one crossover point, there can be also several ones. The mutation can be done also with a sequence mutator that changes a certain sequence of genes into a reverse order. Promising ideas to improve the performance of genetic algorithms are the self-adapting genetic algorithm approach (Hartmann [3]) and the gene bank method (Tyni and Ylinen [5]).

3.2. Priority scheduling procedure

The priority scheduling procedure is given below. It tells how to construct the schedule based on the priorities given by a chromosome. If a population contains *N* chromosomes, there are *N* schedules correspondingly. 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. Create set *E* of all the operations that do not have a predecessor.

2. Select the operation with **the highest priority** from the set *E*. Set the start time of the operation to be the earliest time when the operation could be started with respect to
   - minimum time lags between the operation and its predecessors,
   - ready time constraint of the operation and
   - resource requirements of the operation such that there are enough resources available during the execution of the operation.

   Reduce the resource availabilities respectively.

3. Remove the scheduled operation from set E and add its successors to set E. If set E is empty, stop. Otherwise, go to step 2.

When all the operations are scheduled, the objective function can be calculated based on the properties of the scheduled operations. The priority scheduling procedure above produces a feasible solution (if it exists) with precedence and resource constraints (4) and (7). Depending on the scheduling direction, either ready time (5) or due date constraints (6) 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, in spite 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 are taken into account by using a penalty term in the objective function:

$$J(\mathrm{f}) = J(\mathrm{f}) + \sum_{\forall J_i} w_i \sum_{q=1}^{n_i} \max\{0, f_{iq} - h_{iq}\} \tag{8}$$

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 w_i of job J_i. Correspondingly, in backward scheduling, the penalty terms are used to dealing with the ready time constraints. The backward scheduling objective function is formulated as follows:

$$J(\mathrm{f}) = J(\mathrm{f}) + \sum_{\forall J_i} w_i \sum_{q=1}^{n_i} \max\{0, g_{iq} - s_{iq}\} \tag{9}$$

4. Computational experiments

4.1. Test data

The computational 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 e.g. 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 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 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 $O_{iq\text{-}1}$ (without adding 720).

The scheduling direction is forwards, and the objective is to minimize the makespan, i.e. finish time f_* of the last dummy operation. In the objective function (8), 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 (i.e. the number of operations) 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 after 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. 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.

Table 2.

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.

Preliminary computational results are promising and refer to that good and feasible solutions may be found quickly with the procedure. A reasonable computing time is important so that frequent re-scheduling is possible in the dynamic production environment with external and internal disturbances. The procedure can be applied to quite large real world scheduling problems.

References

[1] Baker A.D., Merchant M.E.: Automatic factories: How will they be controlled?. IEEE Potentials, December 1993: 15-20

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

[3] Hartmann, S.: Self-Adapting Genetic Algorithms with an Application to Project Scheduling. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 506, University of Kiel, Germany (1999)

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

[5] Tyni T., Ylinen, J.: Improving the Performance of Genetic Algorithms with a Gene Bank. In: K. Miettinen, M.M. Mäkelä, J. Toivanen (eds.): Proceedings of EUROGEN99 Short Course on Evolutionary Algorithms in Engineering and Computer Science. Reports of the Mathematical Information Technology No A 2/1999, University of Jyväskylä, Finland, pp. 162-170 (1999)

Contact information

Mr. Olli Kämäräinen (corresponding author)

IC-Parc

Imperial College of Science, Technology and Medicine

William Penney Laboratory

London SW7 2AZ

UK

E-mail: o.kamarainen@ic.ac.uk

Mr. Vesa Ek

TietoEnator

P.O.Box 43

FIN-02131 Espoo

Finland

E-mail: vesa.ek@tietoenator.com

Mr. Kimmo Nieminen

Systems Analysis Laboratory

Helsinki University of Technology

P.O.Box 1100

FIN-02015 HUT

Finland

E-mail: kimmo.nieminen@hut.fi

Prof. Sampo Ruuth

Systems Analysis Laboratory

Helsinki University of Technology

P.O.Box 1100

FIN-02015 HUT

Finland

E-mail: sampo.ruuth@hut.fi

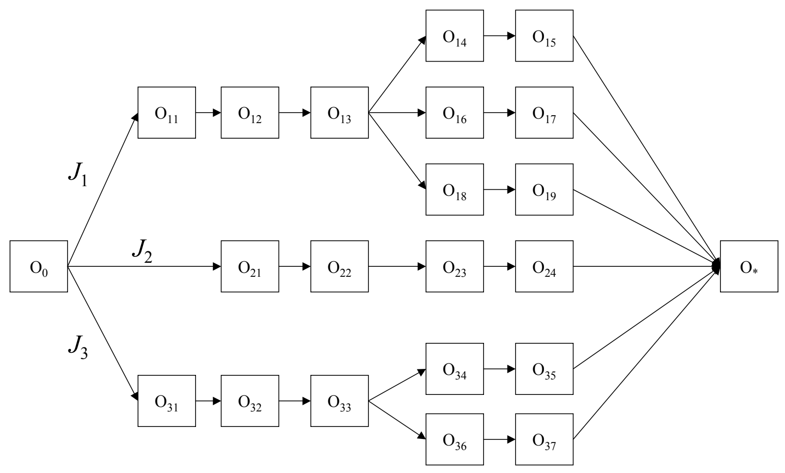

Figure 1. Network of operations

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 1. Computational results

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

Table 2. Approximate computing time per one chromosome