Machine Requirements Planning and Workload Assignment using Genetic Algorithms

B. Porter

Department of Aeronautical, Mechanical and Manufacturing Engineering
University of Salford
Salford M5 4WT
England

K. L. Mak and Y. S. Wong

Department of Industrial and Manufacturing Systems Engineering
The University of Hong Kong
Pokfulam Road
Hong Kong

ABSTRACT

This paper presents a genetic approach to determining the optimal number of machines required in a manufacturing system for meeting a specified production schedule. This use of genetic algorithms is illustrated by solving a typical machine requirements planning problem. Comparison of the respective results obtained by using the proposed approach and a standard mixed-integer programming package shows that the proposed approach is indeed an effective means for optimal manufacturing systems design.

1. Introduction

Optimal machine requirements problem has long been one of the most important research topics in manufacturing systems design. The problem usually involves the optimal selection of manufacturing resources to perform a specified workload. Up to now, numerous attempts have been addressed by a number of researchers (see [1]) to solve the problem. Hayes [2] suggested that the machine requirements planning problem should normally be modelled as a mixed integer programme. However, in the analysis of such a complex problem for a serial flow production system, Hayes et al. [3] pointed out that traditional integer programming algorithms such as branch and bound techniques may prove to be ineffective due to the amount of computational effort required. A dynamic programming approach was then applied instead to determine the optimal solution, but this approach still demands a substantial amount of computational effort.

This paper is concerned with solving machine requirements planning problems in the design of single period, multiple products and multiple machine types manufacturing systems. It is assumed that the manufacturing system under consideration allows (1) the processing time of a piece of work to be different when processed by different machine types in view of the differing capabilities of machines, and (2) a fraction of work to be assigned to machines if necessary. A mathematical model in form of a mixed integer programme is developed for minimizing the total cost of the system. A genetic search algorithm is presented for determining simultaneously the optimal number of machines and the optimal workload assignment for the various types of machines. The application of such an approach and its effectiveness is illustrated by using a numerical example.

2. System Modelling

The manufacturing system considered consists of J different types of machines and each type of machine has its own domain of capacity. However, their domains of capacity are not necessarily disjointed: some of them may overlap. The mathematical model describing the characteristics of such a system is thus governed by the objective function and sets of constraints of the respective forms:

$$\text{minimize} \quad Z = \sum_{j=1}^{J} C_j N_j \qquad (1)$$

$$\text{subject to} \quad \sum_{h=1}^{H} t_{hj} \le E_j N_j \qquad (j = 1,2,...,J) \quad (2)$$

$$\sum_{j=1}^{J} \frac{t_{hj}}{L_{hj}} = 1 \qquad (h = 1,2,...,H) \quad (3)$$

$N_j, t_{hj} \geq 0$ and N_j are integers

$$(h = 1,2,...,H;\ j = 1,2,...,J) \quad (4)$$

where C_j is the cost of employing a machine of type j per period,

E_j is the amount of effective capacity of a machine of type j per period,

N_j is the number of machines of type j selected so as to minimize Z whilst satisfying the constraints (2) and (3),

L_{hj} is the processing time required for the machine of type j to complete the job h,

t_{hj} is the amount of time assigned to machines of type j to perform the job h.

The objective function (1) represents the total cost required per period. Constraints (2) indicate that the total amount of time assigned to each type of machine must be less than or equal to the maximum amount of time available. The ratio t_{hj}/L_{hj} in constraints (3) represents the fraction of job h which is assigned to machines of type j. Since all the jobs must be completed during the planning period, the sum of all these ratios for all types of machine must be equal to 1. Finally, the decision variables N_j (j=1, 2, ..., J) and t_{hj} (h=1, 2, ..., H; j=1, 2, ..., J) are non-negative integer variables and real number variables respectively. Hence, the machine requirements planning problem is formulated as a mixed integer programme which may be solved by using a standard mixed-integer programming package [4] if the problem size is small. However, if the problem size is large, no computationally tractable methodology has yet emerged.

3. Genetic Search Algorithm

Genetic algorithms [5], [6] are newly developed stochastic search techniques which use concepts taken from natural genetics and evolution theory. In this paper, the proposed genetic search algorithm differs from the traditional genetic algorithms in twofold. First, all candidate solutions are directly presented by a floating point representation instead of using a binary coding. It is due to the fact that the size of feasible space of the presented mathematical programme varies in accordance with the system parameters. Hence, it is quite difficult to establish a one-to-one mapping relationship from those feasible points in the space to the chromosome representation. In order to restrict the search on the feasible space only, crossover and mutation operations are also re-designed in the proposed algorithm. Second, a rounding heuristic is used to modify the floating point represented chromosome in order to meet the integer requirements. However, it should be noted that this rounding heuristic is used only on the purpose of evaluating the cost performance of chromosomes. Thus the contents of chromosomes are kept unchanged after the rounding heuristic.

In order to make the proposed algorithm applicable, it is necessary to eliminate all the equality constraints (3) from the presented model. By substituting $t_{hJ} = L_{hJ}(1 - \sum_{j=1}^{J-1} t_{hj}/L_{hj})$ into constraints (2) and re-arrange the constraints, the model is rewritten as:

$$\text{minimize} \quad Z = \sum_{j=1}^{J} C_j N_j \qquad (1)$$

$$\text{subject to} \quad \sum_{h=1}^{H} t_{hj} - E_j N_j \leq 0 \quad (j = 1,2,...,J\text{-}1) \quad (5)$$

$$-\sum_{h=1}^{H}\left[L_{hJ}\sum_{j=1}^{J-1}\frac{t_{hj}}{L_{hj}}\right] - E_J N_J \leq -\sum_{h=1}^{H} L_{hJ} \qquad (6)$$

$$\sum_{j=1}^{J-1} \frac{t_{hj}}{L_{hj}} \leq 1 \qquad (h = 1,2,...,H) \quad (7)$$

$$-t_{hj} \leq 0 \quad (h = 1,2,...,H;\ j = 1,2,...,J\text{-}1) \quad (8)$$

$-N_j \leq 0$ and N_j are integers

$$(j = 1,2,...,J) \quad (9)$$

3.1. Chromosome Structure

Since there are two sets of decision variables namely N_j (j=1,2,...,J) and t_{hj} (h=1,2,...,H; j=1,2,...,J-1) in the system, the combined multi-parameter chromosome, defining any candidate solution, has the form $\tilde{x} = < N >, < t >$ where $< N >$ and $< t >$ are row vectors representing the two sets of decision variables. Hence, the first J bits in a chromosome are used to represent N_j and the remained $H \times (J-1)$ bits are used to represent t_{hj}.

3.2. Initialization Procedure

Chromosomes are seldom feasible if they are just generated randomly. In order to ensure that all chromosomes in the initial population are feasible, an initialization procedure is used. For the sake of simplicity, all constraints in the model are grouped and expressed in the following form:

$$\tilde{A}_k \tilde{x}^T \leq B_k \qquad (k = 1,2,...,K) \quad (10)$$

where K is the total number of constraints in the system,

$\tilde{A}_k$ is a row vector containing the coefficients of the decision variables in the k th constraint (k = 1,2,...,K),

B_k is the constant term involved in the k th constraint (k = 1,2,...,K).

The initialization procedure is simple. First, it is assumed that there is an origin $\tilde{x}_0$ which satisfies the sets of constraints (5)-(9). Starting from this point, a normalized direction vector $\tilde{s}$ is randomly generated. These two vectors are then used to initialize a new point $\tilde{x}$ by:

$$\tilde{x} = \tilde{x}_0 + \theta\tilde{s} \qquad (11)$$

where θ is a randomly generated step size along the direction $\tilde{s}$. For the new point $\tilde{x}$ is feasible, θ must be chosen so that it is less than or equal to its upper bound θ_{max} where θ_{max} is calculated by:

$$\theta_{max} = \min_k \left\{ \frac{B_k - \tilde{A}_k \tilde{x}_0^T}{\tilde{A}_k \tilde{s}^T} \right\} \quad (k = 1,2,...,K) \quad (12)$$

3.3. Rounding Heuristic

The integer requirements of the decision variables N_j are met by using a rounding heuristic to round off the first J decision variables, i.e. N_j (j=1,2,...,J), in a chromosome $\tilde{x}$. In this connection, a straight line is drawn from $\tilde{x}$ to the origin $\tilde{x}_0$. This line is then divided into L equal sections where the value of L is chosen arbitrarily. Obviously, a large L can reduce the computation time while a small L can improve the precision of the results obtained. The outline of the rounding heuristic is given below:

Step 1. Determine a normalized direction vector $\tilde{s}$ from the chromosome $\tilde{x}$ to the origin $\tilde{x}_0$.

Step 2. Divide the straight line joined $\tilde{x}$ and $\tilde{x}_0$ into L equal spaced sections. Let ℓ be an index (ℓ = 1,2,...,L) with an initial value of $\ell = 0$.

Step 3. Generate a new point $\tilde{x}'$ by using $\tilde{x}' = \tilde{x} + \left(\frac{\ell}{L}\right)|\tilde{x}_0 - \tilde{x}|\,\tilde{s}$ and round off the first J decision variables in $\tilde{x}'$ into integers.

Step 4. Check for the feasibility of the rounded $\tilde{x}'$. If it is feasible, go to *Step 6*; if not, go to *Step 5*.

Step 5. Perturb the values of the first J decision variables in $\tilde{x}'$ (one by one) by the values of ±1 and check for the feasibility again. If it is feasible, go to *Step 6*; if not, increase ℓ by 1 and go back to *Step 3*.

Step 6. Evaluate the objective function value of the chromosome $\tilde{x}$ by substituting the rounded $\tilde{x}'$ into the objective function (1) and the rounding is complete.

3.4. Reproduction Operation

Reproduction operation is used to select those potential chromosomes from the current population and place them into a mating pool. In order to make this probabilistic process possible, the cost performance of chromosomes are transformed as fitness values. In this paper, the ranking procedure proposed by Baker [7] is adopted with the selective pressure fixed at 2. In the other words, chromosomes are sorted ascendingly in accordance with their objective function values and then ranks are assigned to them as their fitness values. The fitness value, f, of each chromosome is calculated by $f = f_{max} \times (n - r)/(n - 1)$ where r is the ranking position of the chromosome and f_{max} is the pre-assigned fitness value for the best chromosome. In the selective process, the probability of selecting a chromosome as a parent is defined by $P_{select} = f/\sum_i f_i$ where $\sum_i f_i$ is the sum of fitness values over the whole population. Obviously, a well-performed chromosome will have a greater chance to be selected under this operation.

3.5. Crossover Operation

The exchange of genetic materials (information) of the parent chromosomes is achieved by using the crossover operator. It is a probabilistic operation which is activated occasionally. Its occurrence is controlled by a pre-specified parameter P_{cross}. Each time it occurs, a pair of parent chromosomes $\tilde{x}_1$ and $\tilde{x}_2$ are mated so that a new chromosome called offspring $\tilde{x}$ is generated. To facilitate the calculation, the parent chromosomes are arranged in a manner that the fitness value of $\tilde{x}_1$ is greater than or equal to that of $\tilde{x}_2$. The outline of the crossover operation is given below:

Step 1. Determine the normalized direction vector $\tilde{s}$ from $\tilde{x}_2$ to $\tilde{x}_1$.

Step 2. Generate an offspring chromosome $\tilde{x}$ by using a random step size θ ($0 \le \theta \le \theta_{max}$, see equation (12)) and substitute it into the equation $\tilde{x} = \tilde{x}_2 + \theta\tilde{s}$.

3.6. Mutation Operation

Mutation operation is used to prevent the search process from converging to local optima rapidly. In most linear programming problems, it is found that the

global optimal solutions are always lain on the boundaries of the solution spaces. Hence for the chromosomes which are already lain on the space boundaries, it is a good idea to keep them on those boundaries whenever possible. In this paper, the mutation operation is re-designed so that it can diverge the search to other space regions on the one hand, and keep the chromosomes on the space boundaries on the other hand. This is another probabilistic operation for which its occurrence is controlled by a pre-specified parameter P_{mutate}. Unlike crossover, it is applied to a single chromosome, $\tilde{x}$. The outline of the mutation operation is given below:

Step 1. Set k=1.

Step 2. Check for the condition $\tilde{A}_k \tilde{x}^T = B_k$. If it is satisfied, go to *Step 3*; if not, increase k by 1 and either go back to *Step 2* when $k \le K$ or go to *Step 6* when $k > K$.

Step 3. Perturb the chromosome $\tilde{x}$ by randomly replacing two bits x_i and x_j $(A_{ki}, A_{kj} \neq 0)$ by x_i' and x_j' where

$$x_i' = 0$$

$$x_j' = \frac{A_{kj}x_j + A_{ki}x_i}{A_{kj}}.$$

Step 4. Check for the feasibility of the perturbed $\tilde{x}$. If it is not feasible, go to *Step 5*; otherwise, record the perturbed $\tilde{x}$ and increase k by 1. Then either go back to *Step 2* when $k \le K$ or go to *Step 6* when $k > K$.

Step 5. Determine a normalized direction vector $\tilde{s}$ from the perturbed $\tilde{x}$ to $\tilde{x}$ and shift the perturbed $\tilde{x}$ to a new location $\tilde{x}''$ by $\tilde{x}'' = \text{perturbed } \tilde{x} + \theta\tilde{s}$ where θ is calculated by:

$$\theta = \frac{B_k - \tilde{A}_k(\text{perturbed } \tilde{x})^T}{\tilde{A}_k \tilde{s}^T}$$

Record $\tilde{x}''$ and increase k by 1, then either go back to *Step 2* when $k \le K$ or go to *Step 6* when $k > K$.

Step 6. After the above operations, a number of feasible chromosomes are probably recorded. In this case, the one which has the greatest objective function value will be chosen as the new chromosome $\tilde{x}'$.

4. Numerical Example

The proposed algorithm can be conveniently illustrated by designing a manufacturing system which consists of six types of machine. Twenty jobs are to be processed by the machines during the planning period. The amount of effective capacity and the cost of a machine of type j per period are given in Table 1. The amount of processing time required to perform job h (h=1, 2, ..., 20) by using a machine of type j (j=1, 2, ..., 6) is given in Table 2.

The above data can be substituted into the objective function (1), and the constraints (2) and (3) to derive the mixed-integer programming model. A standard computer package developed by Chang and Sullivan [4] can then be used to determine the optimal number of machines and the optimal assignment of workload to the various types of machines. The reported computation time is 4.4 minutes on a 486DX2-66 PC and the result is presented in Table 3.

The problem is then solved by using the proposed algorithm with the population size of 40. The crossover probability and the mutation probability are 0.8 and 0.3, respectively. The optimal number of machines and the optimal workload assignment to the various types of machines after 50 generations are shown in Table 4. The required computation time is about 3 minutes on the same set of computer and the total cost of the genetically designed system is $32980. It is evident from Tables 3 and 4 that this cost value is exactly the same as that obtained previously by using mixed-integer programming even though their workload assignment plans are slightly different. It also shows that the difference between the machine utilization rates of the respective plans is very small and is mainly due to truncation error. In addition, the amount of time required to produce the optimal solution genetically is only 68.2% of that required by the mixed-integer programming approach.

5. Conclusions

The techniques of using genetic algorithms have been proposed as a means to solve machine requirements planning problems for manufacturing system design. The crossover and mutation operators pursued in the proposed algorithm can limit the search and exploration in the feasible space only and hereby makes the algorithm more efficient. The effectiveness of such approach has been demonstrated by solving genetically a machine requirements planning problem. Comparison of the respective results obtained by using the genetic search algorithm and the mixed integer programming approach clearly shows that the technique proposed in this paper indeed provides a powerful tool for optimal manufacturing systems design.

6. References

[1] D.M. Miller and R.P. Davis, "A dynamic resource allocation model for a machine requirements problem" AIIE Transactions, 10(3), pp. 237-243, 1978.

[2] G.M. Hayes, A Generalized Machine Requirements Planning Algorithm for Serial Flow Machining Systems, M.S. theses, Virginia Polytechnic Institute and State University, 1978.

[3] G.M. Hayes, R.P. Davis and R.A. Wysk, "A dynamic programming approach to machine requirements planning" AIIE Transactions, 13(2), pp. 175-181, 1981.

[4] Y.L. Chang and R.S. Sullivan, QS: Quant Systems Ver 2.0, Prentice-Hall, 1991.

[5] J.H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, 1975.

[6] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, 1989.

[7] J.E. Baker, "Adaptive selection methods for genetic algorithms" Proceedings of the First International Conference on Genetic Algorithms and their Applications, pp. 101-111, 1985.

Table 1. Machine characteristics.

Machine type	1	2	3	4	5	6
Effective capacity (hrs)	78	67	64	82	80	70
Cost per machine ($)	1720	2400	3140	3240	3300	1800

Table 2. Processing time of jobs by using different machine types.

Job	Processing Time (hours)					
No.	m/c1	m/c2	m/c3	m/c4	m/c5	m/c6
1	90	90	-	-	91	-
2	-	98	-	99	-	-
3	66	-	67	70	-	66
4	90	-	86	-	82	-
5	-	26	-	24	-	-
6	-	-	40	43	-	45
7	46	-	-	44	-	-
8	-	56	-	-	59	-
9	-	-	59	-	-	62
10	15	17	-	-	18	-
11	53	-	-	-	-	-
12	-	-	78	-	-	75
13	78	-	77	75	-	-
14	-	-	-	36	-	33
15	62	59	-	-	-	-
16	8	14	-	-	12	-
17	-	-	50	-	47	-
18	-	78	-	80	-	-
19	92	-	91	-	97	-
20	-	-	65	-	-	64

Table 3. The optimal solution by using a standard mixed-integer programming package.

Job	Processing Time Assignment (hours)					
No.	m/c1	m/c2	m/c3	m/c4	m/c5	m/c6
1	90	0	0	0	0	0
2	0	67	0	31.3	0	0
3	66	0	0	0	0	0
4	54	0	0	0	33	0
5	0	0	0	24	0	0
6	0	0	0	0	0	45
7	19	0	0	26.7	0	0
8	0	56	0	0	0	0
9	0	0	0	0	0	62
10	15	0	0	0	0	0
11	53	0	0	0	0	0
12	0	0	0	0	0	75
13	78	0	0	0	0	0
14	0	0	0	0	0	33
15	62	0	0	0	0	0
16	8	0	0	0	0	0
17	0	0	0	0	47	0
18	0	78	0	0	0	0
19	92	0	0	0	0	0
20	0	0	0	0	0	64
Σ	537	201	0	82	80	279
#	7	3	0	1	1	4
%	98.35	100.0	-	100.0	100.0	99.64

Σ = total work load (hrs); # = optimal number of machine types; % = machine utilization (%).

Table 4. The overall best solution by using the proposed algorithm.

Job	Processing Time Assignment (hours)					
No.	m/c1	m/c2	m/c3	m/c4	m/c5	m/c6
1	90	0	0	0	0	0
2	0	67	0	31.3	0	0
3	40.8	0	0	26.7	0	0
4	53.8	0	0	0	33	0
5	0	0	0	24	0	0
6	0	0	0	0	0	45
7	46	0	0	0	0	0
8	0	56	0	0	0	0
9	0	0	0	0	0	62
10	15	0	0	0	0	0
11	53	0	0	0	0	0
12	0	0	0	0	0	75
13	78	0	0	0	0	0
14	0	0	0	0	0	33
15	62	0	0	0	0	0
16	8	0	0	0	0	0
17	0	0	0	0	47	0
18	0	78	0	0	0	0
19	92	0	0	0	0	0
20	0	0	0	0	0	64
Σ	538.6	201	0	82	80	279
#	7	3	0	1	1	4
%	98.65	100.0	-	100.0	100.0	99.64

Σ = total work load (hrs); # = optimal number of machine types; % = machine utilization (%).