A multi-objective lead time control problem in multistage assembly systems using genetic algorithms

Cahit Perkgoz [a], Amir Azaron [b,*], Hideki Katagiri [a], Kosuke Kato [a], Masatoshi Sakawa [a]
[a] Department of Artificial Complex Systems Engineering, Graduate School of Engineering, Hiroshima University, Kagamiyama 1-4-1, Higashi-Hiroshima, Hiroshima, 739-8527 Japan
[b] Cork Constraint Computation Centre, Department of Computer Science, University College Cork, Cork, Ireland

Abstract

In this paper, we develop a multi-objective model to optimally control the lead time of a multistage assembly system, using genetic algorithms. The multistage assembly system is modelled as an open queueing network. It is assumed that the product order arrives according to a Poisson process. In each service station, there is either one or infinite number of servers (machines) with exponentially distributed processing time, in which the service rate (capacity) is controllable. The optimal service control is decided at the beginning of the time horizon. The transport times between the service stations are independent random variables with generalized Erlang distributions. The problem is formulated as a multi-objective optimal control problem that involves four conflicting objective functions. The objective functions are the total operating costs of the system per period (to be minimized), the average lead time (min), the variance of the lead time (min) and the probability that the manufacturing lead time does not exceed a certain threshold (max). Finally, we apply a genetic algorithm with double strings using continuous relaxation based on reference solution updating (GADSCRRSU) to solve this multi-objective problem, using goal attainment formulation. The results are also compared against the results of a discrete-time approximation technique to show the efficiency of the proposed genetic algorithm approach.

Keywords: Queueing; Genetic algorithms; Multiple objective programming; Production

1. Introduction

Over the last decade, manufacturing strategies have focused on speed of response to customer as much as cost and quality for competitive advantage. This means reducing both the length and the variability of the manufacturing lead times. Short lead times are critical to win customer orders for engineer-to-order and make-to-order companies supplying capital goods. These products are often complex assemblies with many stages of manufacture and assembly. Providing competitive delivery lead times and managing to achieve a reliable delivery performance are typically as important as competitive prices.

The multistage assembly system is modelled as an open queueing network, where each service station settled in a node of the network represents a manufacturing or assembly operation. It is assumed that only one type of product is produced by the system. Each product consists of a number of separate raw parts, which should be processed and assembled to each other. Each separate part of the product enters the production system according to a Poisson process and goes to the first service station in its routing sequence of manufacturing operations.

[*] Corresponding author. *E-mail address*: *E-mail address*: a.azaron@4c.ucc.ie (A. Azaron).

After completing the manufacturing operations of the separate parts, they are assembled to each other and after passing some other manufacturing and assembly operations, the final product leaves the system in its finished form.

In each service station, there is either one or infinite number of servers with exponential distribution of processing time. In application, if the product should wait for starting the service, in front of the station, it can be represented as an *M/M/1* queueing system. Otherwise, if there are enough servers, in which the product does not need to wait in queue, we can represent it as an $M/M/\infty$ queueing system.

The role of transport times between the service stations, which may be much greater than the processing times, are also considered to compute and then optimize the manufacturing lead time in dynamic multistage assembly systems. An implicit hypothesis in the literature is that transit time in buffers is null, *i.e.*, a part which leaves a machine is supposed to be instantaneously available for the next machine. The transport times between the service stations are assumed to be independent random variables with generalized Erlang distributions.

The time spent in a service station would be equal to the processing time plus waiting in the queue in front of the service station. Therefore, the time spend by a finished product in the system, called manufacturing lead time, would be equal to the length of the longest path of the queueing network whose arc lengths are the transport times between the service stations. We can obtain the distribution function of the manufacturing lead time by computing the distribution function of the longest path length in the queueing network.

Yano [25] considered stochastic lead time in a simple two level assembly system with different processing time distributions including Poisson and negative binomial. Cheng and Gupta [4] commented that most analytical studies are limited to small problems. Song *et. al.* [23] developed an approximate method to obtain the distribution of product completion time by decomposing the complex product structures of multistage assemblies into two-stage subsystem.

The analytical methods above consider the manufacturing and assembly processing times as independent random variables and ignore their dependence on the arrival and service rates of jobs at various stages in the manufacturing process. The time spent in a queue will be longer for congested service stations than for little used stations. Therefore, the time spent waiting in queues in front of service stations should be considered in order to compute the manufacturing lead time.

The open queueing networks are widely used for modelling manufacturing systems, see Papadopoulos and Heavey [16]. The lead time analysis in dynamic job shops by modelling those as open queueing networks was studied by Kapadia and Hsi [11], Shanthikumar and Sumita [19], Haskose *et. al.* [9] and Vandaele *et. al.* [24]. However, these works do not include assembly processes.

Harrison [8], in a primarily theoretical study, introduced a queueing theoretical model of an assembly operation. He established stability conditions for an assembly queue with renewal and mutually independent arrival streams and a single server. An approximate analysis of the assembly-like queue with finite queues of equal length and under symmetric load has been presented by Lipper and Sengupta [14]. Hemachandra and Eedupuganti [10] considered a model of a system with two finite capacity assembly lines and a single join operation and presented an approach for computing the performance measures in the system. Gold [5] considered a model corresponds to an assembly-like queue with two input streams, in which the assembly is instantaneous, and focused on the state probabilities and expectation of minimum and maximum of the two input queues. Ramachandran and Delen [17] analyzed the kitting process (a kit is a set of parts which are all needed to perform the assembly) as of a stochastic assembly system by treating it as an assembly-like queue. Specially, they investigated the dynamics involved in a simple kitting process where two independent input streams feed into an assembly process.

All above papers discuss about single stage assembly systems. Moreover, most of them are either based on the finite buffer capacity assumption or consider that there are no external arrivals that corresponds to orders or assume that assembly is instantaneous.

Azaron *et al.* [1] relaxed these restricted assumptions and developed a discrete-time approximation technique to optimally control the service rates (capacities) of the manufacturing and the assembly operations in dynamic multistage assembly systems, in which the average lead time, the variance of the lead time and the total operating costs of the system per period are minimized.

In this paper, we extend the work of Azaron *et al.* [1] in the following directions. First, the probability that the manufacturing lead time does not exceed a certain threshold, which is one of the most important criteria in the stochastic programming concept and has not been considered in [1], is considered as one of the objectives of the final multi-objective problem. Second, the transport times are assumed to follow generalized Erlang distributions, instead of exponential distributions in [1]. Third, the number of servers can be either one or infinite, while in [1] there should be only one server in each service station. Forth, it is proved that solving the resulting multi-objective problem using the standard optimal control tools is impossible, and a genetic algorithm is applied to solve the problem, accordingly. Finally, we solve some illustrative examples and compare the results against the results of the discrete-time approximation technique, developed by Azaron *et al.* [1], to show the efficiency of the proposed genetic algorithm approach. In this paper, the optimal service control is decided at the beginning of the time horizon like [1].

For the problem concerned in this paper, as a general-purpose solution method for discrete nonlinear programming problems, in order to consider the nonlinearity of problems and to cope with large-scale problems, we propose the usage of GADSCRRSU, which is a direct extension of genetic algorithms with double strings based on a reference solution updating (GADSRSU) for linear 0-1 programming problems, see Sakawa and Kato [18].

The remainder of this paper is organized in the following way. In Section 2, we explain the structure of dynamic multistage assembly systems. In Section 3, the lead time distribution in dynamic multistage assembly systems is obtained. In Section 4, we present the multi-objective lead time control problem. In Section 5, we explain about GADSCRRSU. Section 6 presents the computational experiments, and finally we draw the conclusion of the paper, in Section 7.

2. Dynamic multistage assembly systems

In our methodology, the following assumptions will be made.

1. Each separate part of the product enters the production system according to a Poisson process with rate λ (the demand rate for the final product).
2. Only one type of product is produced.
3. Each service station with only one incoming arc indicates a manufacturing station.
4. Each service station with more than one incoming arcs indicates an assembly station.
5. After a separate part arrives at the system, it goes directly to a manufacturing station for its first manufacturing operation. If there are parts for being processed, it queues up.

6. After completion of processing at a manufacturing station, it goes to another manufacturing station to be processed in its routing sequence of manufacturing operations.
7. After completing the manufacturing operations of each separate part, it is assembled to other separate parts in an assembly station.
8. The product leaves the system in its finished form from the sink node of the queueing network.
9. Each part has characteristics, which are statistically independent of other parts.
10. Each service station consists of either one or infinite number of servers.
11. Processing times of manufacturing and assembly operations are exponentially distributed (including set up times on the service station).
12. The processing time at each service station is independent of preceding processing times.
13. There are no interruptions due to breakdowns, maintenance, or other such cases.
14. Service discipline is based on FIFO.
15. All inter-station buffers are infinite.
16. The transport times between the service stations are independent random variables with generalized Erlang distributions.
17. The queueing network is in the steady-state.
18. Capacity is controlled through the service rate at each node.
19. Service rates are stepwise variables.
20. Operating cost of each service station per period is an increasing function of its service rate.
21. Total number of service stations settled in the nodes of the queueing network is equal to *n*.

Having completed one assembly, the server immediately begins another if at least one input item of each separate part is available. Otherwise, one or more parts have to wait for the last one to arrive (synchronization loss). The reader will recognize that each assembly station is a multi-input generalization of *M/M/1* queue. Its salient feature is a very special type of batch servicing, each batch containing exactly one customer of each part. Harrison [8] proved that the arrival pattern at the assembly node would not be renewal. Therefore, developing an analytical method to match the synchronization loss and non-renewal arrival streams in order to find the lead time distribution in multistage assembly systems is impossible, and we restrict our attention to an approximation one.

Clearly, the arrival process to the manufacturing stations prior to an assembly station is Poisson with the rate of λ. It is shown that the arrival of kits at each assembly node can be considered as Poisson with the rate of λ, as long as the parts arrive as a Poisson process of rate λ, see [1] for details.

Every two nodes of the queueing network associated with a dynamic multistage assembly system are connected by at most one directed path, *i.e.*, the network is a tree, and consequently the waiting times in the service stations are independent, see Lemoine [13].

3. Lead time distribution in dynamic multistage assembly systems

The main steps of our proposed method are as follows:

Step 1. Compute the density function of time spent (processing time plus waiting time in queue) in each service station.

Step 1.1. If there is one machine in the *i*th service station, then the density function of time spent in this *M/M/1* queueing system is

$$w_i(t) = (\mu_i - \lambda)e^{-(\mu_i - \lambda)t} \quad t>0 \qquad (1)$$

where λ and μ_i are the arrival rate and the service rate of this queueing system, respectively. Therefore, the density function of time spent in the *i*th service station would be exponential with parameter *(* μ_i *-* λ *)*.

Step 1.2. If there are infinite number of machines in the *i*th service station, then the time spent in this *M/M/*∞ queueing system will be exponentially distributed with parameter μ_i, because there is no queue.

Step 2. Transform the queueing network into an equivalent stochastic network by replacing each node including a service station with a stochastic arc whose length is equal to the time spent in the particular service station.

Let's explain how to replace node *k* in the queueing network, which includes a queueing system, with a stochastic arc. Assume that $b_1, b_2, \ldots, b_n$ are the incoming arcs to this node and $d_1, d_2, \ldots, d_m$ are the outgoing arcs from it. Then, we substitute this node by arc *(k', k'')*, whose length is equal to the time spent in the corresponding queueing system. Furthermore, all arcs b_i for $i=1,\ldots,n$ end up with *k'* while all arcs d_j for $j=1,\ldots,m$ start from node *k''*. The indicated process is opposite of the absorption an edge *e* in a graph *G* in graph theory (*G.e*), see Azaron and Modarres [2] for more details. After transforming all such nodes to the proper stochastic arcs, the queueing network is transformed into an equivalent stochastic network.

Step 3. Transform the original stochastic network, obtained in step 2, into a new one with exponentially distributed arc lengths.

For constructing this new network, we use the idea that if the length of each arc a in the original stochastic network, corresponding with a transport time in the original queueing network, is distributed according to a generalized Erlang distribution of order n_a and the infinitesimal generator matrix G_a as:

$$G_a = \begin{bmatrix} -\lambda_{a1} & \lambda_{a1} & 0 & \ldots & 0 & 0 \\ 0 & -\lambda_{a2} & \lambda_{a2} & \ldots & 0 & 0 \\ . & . & . & \ldots & . & . \\ 0 & 0 & 0 & \ldots & -\lambda_{a n_a} & \lambda_{a n_a} \\ 0 & 0 & 0 & \ldots & 0 & 0 \end{bmatrix},$$

it can be decomposed into n_a exponential serial arcs with the parameters $\lambda_{a1}, \lambda_{a2}, \ldots, \lambda_{a n_a}$. Then, this generalized Erlang arc is substituted with n_a series of exponential arcs with the parameters $\lambda_{a1}, \lambda_{a2}, \ldots, \lambda_{a n_a}$. After substituting all such generalized Erlang arcs with the proper exponential serial arcs, the original stochastic network is transformed into a new one with exponentially distributed arc lengths.

Step 4. Compute the distribution function of longest path in the new stochastic network obtained in step 3.

Any analytical method dealing with the computation of longest path length distribution in stochastic networks with exponentially distributed arc lengths can be used in step 4. We use the method of Kulkarni and Adlakha [12] in this step, because this method is an analytical one, simple, easy to implement on a computer and computationally stable.

Let *G=(V,A)* be the new stochastic network, in which *V* represents the set of nodes and *A* represents the set of arcs or operations of the dynamic assembly system after the transformation. The source and sink nodes are denoted by *s* and *y*, respectively. Length of arc $a \in A$ is an exponentially distributed random variable with parameter γ_a. For $a \in A$, let $\alpha(a)$ and $\beta(a)$ be the starting and ending nodes of arc *a*, respectively.

Definition 1: Let *I(v)* and *O(v)* be the sets of arcs ending and starting at node *v*, respectively, which are defined as follows:

$$I(v) = \{a \in A : \beta(a) = v\} \quad (v \in V) \tag{2}$$

$$O(v) = \{a \in A : \alpha(a) = v\} \quad (v \in V) \tag{3}$$

Definition 2: If $X \subset V$ such that $s \in X$ and $y \in \overline{X}$ *=V-X,* then an *(s,y)* cut is defined as:

$$(X, \overline{X}) = \{a \in A : \alpha(a) \in X, \beta(a) \in \overline{X}\} \tag{4}$$

An *(s,y)* cut $(X, \overline{X})$ is called a uniformly directed cut (UDC)**,** if $(\overline{X}, X)$ is empty.

Definition 3: Let $D = E \cup F$ be a uniformly directed cut (UDC) of a network. Then, it is called an admissible 2-partition, if for any $a \in F$, we have $I(\beta(a)) \not\subset F$.

Definition 4: Each operation at time *t* can be in one of the active**,** dormant or idle states, which are defined as follows:

i. Active: an operation is active at time *t,* if it is being executed at time *t.*

ii. Dormant: an operation *a* is dormant at time *t,* if it is finished but there is at least one unfinished operation in $I(\beta(a))$. If an operation is dormant at time *t,* then its successor operations in $O(\beta(a))$ cannot begin.

iii. Idle**:** an operation is idle at time *t,* if it is neither active nor dormant at time *t.*

The set of active and dormant states are denoted by *Y(t), Z(t),* respectively, and *X(t)=(Y(t),Z(t)).* Let *S* denote the set of all admissible 2-partition cuts of the network, and $\overline{S} = S \cup \{(\phi,\phi)\}$. Note that *X(t)=* (ϕ,ϕ) implies that *Y(t)=* ϕ and *Z(t)=* ϕ, *i.e.* all operations are idle at time *t* and hence the final product is completed by time *t.*

It is proven that *{X(t),t≥0}* is a continuous-time Markov process with state space $\overline{S}$. The elements of the infinitesimal generator matrix $Q = [q\{(E,F),(E',F')\}]$, *(E,F)* and $(E',F') \in \overline{S}$, where *E* and *F* include active and dormant operations of a UDC, respectively, are calculated as follows (refer to [12] for details):

$$q\{(E,F),(E',F')\} = \begin{cases} \gamma_a \quad \text{if} \quad a \in E, I(\beta(a)) \not\subset F \cup \{a\}, E' = E - \{a\}, F' = F \cup \{a\}; & (5) \\ \gamma_a \quad \begin{array}{l} \text{if} \quad a \in E, I(\beta(a)) \subset F \cup \{a\}, E' = (E - \{a\}) \cup O(\beta(a)), \\ F' = F - I(\beta(a)); \end{array} & (6) \\ -\sum_{a \in E} \gamma_a \quad \text{if} \quad E' = E, F' = F; & (7) \\ 0 \qquad \textit{otherwise.} & (8) \end{cases}$$

{X(t),t ≥ 0} is a finite-state absorbing continuous-time Markov process and since $q\{(\phi,\phi),(\phi,\phi)\} = 0$, it is concluded that this state is an absorbing one and obviously the other states are transient. Furthermore, we number the states in $\overline{S}$ such this *Q* matrix be an upper triangular one. We assume that the states are numbered *1,2,…,N=* $|\overline{S}|$. State *1* is the initial state, namely *X(t)=* $(O(s),\phi)$, and state *N* is the absorbing state, namely *X(t)=* (ϕ,ϕ).

Let T represent the length of the longest path in the network, or the manufacturing lead time. Clearly, $T=\min\{t>0: X(t)=N/X(0)=1\}$. Thus, T is the time until $\{X(t),t\geq 0\}$ gets absorbed in the final state starting from state 1.

Chapman-Kolmogorov backward equations can be applied to compute $F(t)=P\{T\leq t\}$. If we define:

$$P_i(t)=P\{X(t)=N/X(0)=i\} \quad i=1,2,\ldots,N \tag{9}$$

then, $F(t)=P_1(t)$.

The system of linear differential equations for the vector $P(t)=[P_1(t),P_2(t),\ldots,P_N(t)]^T$ is given by

$$\begin{aligned}&\dot{P}(t)=Q.P(t)\\&P(0)=[0,0,\ldots,1]^T\end{aligned} \tag{10}$$

4. Multi-objective lead time control problem

In this section, we develop an analytical model to optimally control the service rates of the service stations. In fact, we may increase the service rates of the manufacturing and assembly stations by allocating more resources. In that case, the average manufacturing lead time will be decreased. However, clearly it causes the total operating costs of the system per period to be increased, accordingly. Consequently, an appropriate trade-off between the average lead time and cost is required. The variance of lead time should also be considered in the model, because when we only focus on such mean time, the service rates may be non optimal if the lead time substantially varies because of randomness. Interpretation of the variance of lead time is very difficult in application and consequently we need another proper deterministic objective associated with the minimization of lead time, which has a random nature. This new objective would be the probability that the manufacturing lead time does not exceed a certain threshold as the fourth objective of our multi-objective formulation.

To achieve the above-mentioned goals, we develop a multi-objective problem, in which four objectives are sought simultaneously, minimizing the total operating costs of the system per period, minimizing the average lead time, minimizing the variance of lead time and also maximizing the probability that the manufacturing lead time does not exceed a given threshold.

The operating cost of the ith service station per period is assumed to be an increasing function $C_i(\mu_i)$ of its service rate μ_i. Therefore, C or the total operating costs of the system per period is given by

$$C=\sum_{i=1}^{n} C_i(\mu_i) \tag{11}$$

We focus on the case where capacity is available in discrete options, such as when machines, workers, or shits are to be added. Capacity is controlled through the service rate at each node. Therefore, each service rate should be selected from a related discrete set of choices.

Considering S_i as the set of choices for the ith service rate $(\mu_i \in S_i)$, the infinitesimal generator matrix Q is not constant, rather it would be a function of the control vectors $\mu=[\mu_1,\mu_2,\ldots,\mu_n]^T$. Therefore, the non-linear dynamic model is

$$\begin{aligned}&\dot{P}(t)=Q(\mu).P(t)\\&P_i(0)=0 \quad i=1,2,\ldots,N-1\\&P_N(t)=1\end{aligned} \tag{12}$$

Representing *B* as the set of nodes including *M/M/1* service stations and *C* as the set of nodes including *M/M/*∞ service stations in the original multistage assembly system, the relations (13) should be satisfied to exist the response in the steady-state.

$$\mu_i > \lambda \quad i \in B$$
$$\mu_i > 0 \quad i \in C \tag{13}$$

We do not have such constraints in the mathematical programming. Therefore, we use the constraints (14) instead of the above constraints in the final multi-objective problem, assuming ε as a small quantity which should approach zero.

$$\mu_i \geq \lambda + \varepsilon \quad i \in B$$
$$\mu_i \geq \varepsilon \quad i \in C \tag{14}$$

Accordingly, the appropriate multi-objective optimal control problem is

$$Min \quad f_1(\mu) = C_i(\mu_i)$$
$$Min \quad f_2(\mu) = \int_0^\infty (1 - P_1(t))dt$$
$$Min \quad f_3(\mu) = \int_0^\infty t^2 \dot{P}_1(t)dt - \left[\int_0^\infty t\, \dot{P}_1(t)dt\right]^2$$
$$Max \quad f_4(\mu) = P_1(u)$$

s.t:

$$\dot{P}(t) = Q(\mu).P(t)$$
$$P_i(0) = 0 \quad i = 1,2,\ldots,N-1$$
$$P_N(t) = 1$$
$$\mu_i \geq \lambda + \varepsilon \quad i \in B$$
$$\mu_i \geq \varepsilon \quad i \in C$$
$$\mu_i \in S_i \quad i = 1,2,\ldots,n \tag{15}$$

A possible approach to solving (15) to optimality is to use the Maximum Principle (see Sethi [22] for details). For simplicity, consider solving the problem with only one of the objective functions, $f_2(\mu) = \int_0^\infty (1 - P_1(t))dt$.

Consider Λ as the set of allowable controls consisting of all constraints except the constraints representing the dynamic model ($\mu \in \Lambda$), and *N*-vector $\lambda(t)$ as the adjoint vector function. Thus, Hamiltonian function would be

$$H(\lambda(t), P(t), \mu) = \lambda(t)^T Q(\mu).P(t) + 1 - P_1(t) \tag{16}$$

Now, we write the adjoint equations and the terminal conditions, which are

$$-\dot{\lambda}(t)^T = \lambda(t)^T.Q(\mu) + [-1,0,\ldots,0]$$
$$\lambda(T)^T = 0, \quad T \to \infty \tag{17}$$

If we could compute $\lambda(t)$ from (17), then we would be able to minimize the Hamiltonian function subject to $\mu \in \Lambda$ in order to get the optimal control μ^* and to solve the problem, optimally. Unfortunately, the adjoint equations (17) are dependent on the unknown control vector μ and therefore they cannot be solved directly.

If we could also minimize the Hamiltonian function (16), subject to $\mu \in \Lambda$, for an optimal control function in closed form as $\mu^* = f(P^*(t), \lambda^*(t))$, then we would be able to substitute this into the state equations, $\dot{P}(t)=Q(\mu).P(t),\ P(0) = [0,0,...,1]^T$, and adjoint equations (17) to get a set of differential equations, which is a two-point boundary value problem. Unfortunately, we cannot obtain μ^* by differentiating H with respect to μ , because μ is a discrete vector and consequently μ^* cannot be obtained in a closed form.

According to these points, it is impossible to solve the optimal control problem (15), optimally, even in the restricted case of a single objective problem. Relatively few optimal control problems can be solved optimally. Therefore, we try to solve this problem, using genetic algorithms, considering the goal attainment formulation.

4.1. Goal attainment method

Goal attainment method requires setting up a goal and weight, b_j and c_j ($c_i \geq 0$) for $j=1,2,3,4$, for the four objective functions. The c_j relate the relative under-attainment of the b_j. For under-attainment of the goals, a smaller c_j is associated with the more important objectives. c_j, $j=1,2,3,4$, are generally normalized so that $\sum_{i=1}^{4} c_i = 1$.

The appropriate goal attainment formulation of the discrete lead time control problem leads to:

$$
\begin{aligned}
&Min\ \ z \\
&s.t.: \\
&\sum_{i=1}^{n} C_i(\mu_i) - c_1 z \leq b_1 \\
&\int_0^\infty t\, \dot{P}_1(t)dt - c_2 z \leq b_2 \\
&\int_0^\infty t^2\, \dot{P}_1(t)dt - \left[\int_0^\infty t\, \dot{P}_1(t)dt\right]^2 - c_3 z \leq b_3 \\
&P_1(u) + c_4 z \geq b_4 \\
&\dot{P}(t)=Q(\mu).P(t) \\
&P_i(0)=0 \qquad i=1,2,\ldots,N-1 \\
&P_N(t)=1 \\
&\mu_i \geq \lambda + \varepsilon \qquad i \in B \\
&\mu_i \geq \varepsilon \qquad i \in C \\
&\mu_i \in S_i \qquad i=1,2,\ldots,n \\
&z \geq 0
\end{aligned}
\qquad (18)
$$

Lemma 1. If μ^* is Pareto-optimal, then there exists a c, b pair such that μ^* is an optimal solution to the optimization problem (18). □

The optimal solution using this formulation is fairly sensitive to b and c. Depending upon the values for b, it is possible that c does not appreciably influence the optimal solution. Instead, the optimal solution can be determined by the nearest Pareto-optimal solution from *b*. This might require that *c* be varied parametrically to generate a set of Pareto-optimal solutions.

5. A genetic algorithm with double strings using continuous relaxation based on reference solution updating (GADSCRRSU)

In this section, we mention GADSCRRSU proposed as a general solution method for discrete programming problems defined as (19).

$$\begin{aligned}
&Min \quad f(\mu)\\
&s.t.:\\
&A\mu \le b\\
&\mu_j \in \{0,1,\dots,\nu_j\} \quad j=1,\dots,n
\end{aligned} \tag{19}$$

where μ is an *n* dimensional discrete decision variable vector.

In order to have the same form given in (19), we reformulate the problem (18), by combining the objective functions and the state equations. Considering $\mu=[\mu_1,\mu_2,\dots,\mu_n]^T$ as the decision vector, the appropriate min-max problem is obtained as, in which $f_i(\mu),\ i=1,2,3,4$, are the same as (15):

$$\begin{aligned}
&Min \quad f(\mu)=Max\{z_1(\mu),z_2(\mu),z_3(\mu),z_4(\mu)\}\\
&s.t.:\\
&\mu_j \in S_j \qquad j=1,2,\dots,n
\end{aligned} \tag{20}$$

where

$$z_1(\lambda)=\frac{f_1(\mu)-b_1}{c_1},$$

$$z_2(\lambda)=\frac{f_2(\mu)-b_2}{c_2},$$

$$z_3(\lambda)=\frac{f_3(\mu)-b_3}{c_3},$$

$$z_4(\lambda)=\frac{b_4-f_4(\mu)}{c_4},$$

and

$$\dot{P}(t)=Q(\mu).P(t),\ P(0)=[0,0,\dots,1]^T. \tag{21}$$

It should be noted that in our computer program, $P_1(t)$ is obtained by solving the system of differential equations (21), analytically, and then the average and the variance of lead time are computed, numerically. In the problem (20), the linear constraints ($A\mu \le b$) of problem (19) are considered, implicitly. The only restriction that we have in this problem is that the elements of μ vector (decision variables) are selected from the given sets.

Before going into details of the genetic algorithm that will be used in this paper, the definitions of the parameters of the genetic algorithm should be explained. *N* is the number of population of the genetic algorithm. At the beginning, *N* individuals are created as the population. *G* is the generation gap, which is used in the crossover operation. After the generation of offspring from the current population, the number of individuals that will be added into the population in the next iteration is selected according to *G* from the offspring population. The probability of crossover p_c is also used in crossover operation to determine the probability of applying the crossover operator to each individual. The probability of mutation p_m and the probability of inversion p_i are used in the same way in order to determine the mutation operator and the inversion operator are going to be applied to an individual or not, respectively. As it is well known, genetic algorithms are iterative search methods, where the number of iterations should be specified in the algorithm. In this paper, we use I_{min} and I_{max}, which are the minimum and the maximum number of iterations that would be performed. For example, the process will not stop before I_{min} number of iterations and will stop after performing I_{max} number of iterations. The mutation operator is applied according to Gaussian distribution and the uniform distribution. A parameter *R* is used for choosing the type of mutation operator that will be applied to an individual.

5.1. Individual representation

The individual representation by double strings, shown in Figure 1, is adopted in GADSCRRSU.

Indices	$s(1)$	$s(2)$	$\cdots$	$s(j)$	$\cdots$	$s(n)$	$= \mathbf{s}$
Values	$g_{s(1)}$	$g_{s(2)}$	$\cdots$	$g_{s(j)}$	$\cdots$	$g_{s(n)}$	

Figure 1. Double strings representation

In this figure *s*(*j*), *j*=1, ..., *n,* is the index of the *j*th element of the double strings and $g_{s(j)} \in S_{s(j)}$, *j*=1, ..., *n,* is the integer value of the corresponding element, respectively. For example an individual $\mu = \{\mu_1, \mu_2, \mu_3, \mu_4, \mu_5, \mu_6\} = \{4,8,5,9,9,2\}$ may be represented as in Figure 2 at an iteration during the solution process for *n* = 6.

Indices	4	2	1	6	3	5	$= \mu$
Values	9	8	4	6	5	9	

Figure 2. Double strings representation

5.2. Decoding algorithm

In this paper, we use the framework of Sakawa [19] to construct a decoding algorithm of double strings to solve the discrete programming problem (20). In the algorithm, a feasible solution $\hat{\mu}$ of the continuous relaxation problem, called a reference solution, is used as the origin of decoding. The reference solution updating procedure is also adopted, see Sakawa [19] for details, since the solutions obtained by the decoding algorithm using a reference solution tend to concentrate around the reference solution.

Decoding algorithm using continuous programming relaxation

Step 1. Let $j = 1$ and $sum_i = 0, i = 1,...r.$ r is the number of constraints in the problem.

Step 2. If $\hat{\mu}_{s(j)} > 0$, proceed to step 3. Otherwise, *i.e.*, if $\hat{\mu}_{s(j)} = 0,$ let $j = j + 1$ and go to step 5.

Step 3. Let $a_{is(j)}$ denote the $(i, s(j))$ element of the coefficient matrix *A*. Then, $\mu_{s(j)}$ is determined as:

$$\mu_{s(j)} = \min\left(\min_{i=1,\dots,r} \left\lfloor \frac{b_i - sum_i}{a_{is(j)}} \right\rfloor, g_{s(j)} \right),$$

where $a_{is(j)} \neq 0$.

Step 4. Let $sum_i = sum_i + a_{is(j)}\mu_{s(j)},$ $i = 1,...r$ and $j = j + 1$.

Step 5. If $j > n$, proceed to step 6. Otherwise, return to step 2.

Step 6. Let $j = 1$.

Step 7. If $\hat{\mu}_{s(j)} = 0,$ proceed to step 8. Otherwise, *i.e.*, if $\hat{\mu}_{s(j)} > 0$, let $j = j + 1$ and go to step 10.

Step 8. Let $a_{is(j)}$ denote the $(i, s(j))$ element of the coefficient matrix *A*. Then, $\mu_{s(j)}$ is determined as:

$$\mu_{s(j)} = \min\left(\min_{i=1,\dots,r} \left\lfloor \frac{b_i - sum_i}{a_{is(j)}} \right\rfloor, g_{s(j)} \right),$$

where $a_{is(j)} \neq 0$.

Step 9. Let $sum_i = sum_i + a_{is(j)}\mu_{s(j)},$ $i = 1,...r$ and $j = j + 1$.

Step 10. If $j > n$, proceed to step 6. Otherwise, return to step 7.

5.3. Usage of continuous relaxation

In large scale problems, we need some schemes such as the generation of individuals near the optimal solution, the restriction of the search space to a promising region and so forth, in order to find an approximate optimal solution with high accuracy in reasonable time. From this point of view, the information about an optimal solution to the corresponding continuous relaxation problem

$$Min\ \ z = f(\mu)$$

s.t:

$$0 \le \mu_j \le \nu_j, \quad j = 1, \ldots, n \tag{22}$$

is used in the generation of the initial population and the mutation.

5.4. Genetic Operators

5.4.1. Reproduction

For genetic algorithms, various reproduction methods, see for example Goldberg [6] and Michalewicz [15], have been proposed such as; ranking selection, elitist ranking selection, expected value selection, elitist expected value selection, roulette wheel selection, and elitist roulette wheel selection. In this paper, elitist expected value selection, which is a combination of elitist preserving selection and expected value selection, is adopted as a reproduction operator.

Elitist preserving selection: One or more individuals with the largest fitness up to the current population is unconditionally preserved in the next generation.

Expected value selection: Let N denote the number of individuals in the population. The expected value of the number of the ith individual $\boldsymbol{s}_i$ in the next population is calculated as:

$$N_i = \frac{f(\mathbf{s}_i)}{\sum_{i=1}^{N} f(\mathbf{s}_i)} * N. \tag{23}$$

In expected value selection, the integral part of (23) denotes the definite number of individuals $\boldsymbol{s}_i$ preserved in the next population. Using the fractional part of (23), the probability to preserve $\boldsymbol{s}_i$ in the next population is determined by

$$\frac{N_i - \lfloor N_i \rfloor}{\sum_{i=1}^{N} (N_i - \lfloor N_i \rfloor)}. \tag{24}$$

5.4.2. Crossover

If a single-point crossover or multipoint crossover is directly applied to individuals of double string type, the kth element of an offspring may take the same number of the k'th element. In order to avoid this, a crossover method called partially matched crossover (PMX) was proposed by Goldberg and Lingle [7] and was modified to be suitable for double strings by Sakawa *et al.* [20]. The PMX for double strings can be described as follows:

Partially Matched Crossover (PMX) for double strings

Step 0. Set $w = 1$.

Step 1. Choose X and Y as parent individuals. Then, let $X' = X$ and $Y' = Y$.

Step 2. Generate a real random number rand() in [0,1]. For a given crossover rate p_c, if rand()$\le p_c$, then go to step 3. Otherwise, go to step 8.

Step 3. Choose two crossover points h, k $(h \ne k)$ from $\{1,2,\ldots,n\}$ at random. Then, set $l = h$. First, perform operations in step 4 through 6 for X' and Y.

Step 4. Let $j = ((l-1)\%n)+1$ ($p\%q$ is defined as the remainder when an integer *p* is divided by an integer *q*). After finding j' such that $s_Y(j) = s_{X'}(j')$, interchange $(s_{X'}(j), g_{s_{X'}(j)})^T$ with $(s_{X'}(j'), g_{s_{X'}(j')})^T$. Furthermore, set $l = l+1$, and go to step 5.

Step 5. 1) If $h < k$ and $l > k$, then go to step 6. If $h < k$ and $l \le k$ then return to step 4. 2) If $h > k$ and $l > (k+n)$, then go to step 6. If $h > k$ and $l \le (k+n)$, then return to step 4.

Step 6. 1) If $h < k$ let $g_{s_{X'}(j)} = g_{s_Y(j')}$ for all *j* such that $h \le j \le k$, and go to step 7. 2) If $h > k$, let $g_{s_{X'}(j)} = g_{s_Y(j')}$ for all *j* such that $1 \le j \le k$ and $h \le j \le n$, then go to step 7.

Step 7. Carry out the same operations as in steps 4 through 6 for Y' and *X*.

Step 8: Preserve X' and Y' as the offspring of *X* and *Y*.

Step 9. If $w < N$, set $w = w+1$ and return to step 1. Otherwise, go to step 10.

Step 10. Choose $N*G$ individuals from $2*N$ preserved individuals randomly and replace $N*G$ individuals of the current population consisting of *N* individuals with the $N*G$ chosen individuals. Here, *G* is a constant called generation gap.

5.4.3. Mutation and Inversion

The procedures of mutation and inversion for double strings are summarized as follows:

Mutation for double strings

Step 0. Let $w = 1$.

Step 1. Let $j = 1$.

Step 2. If a real random number rand () in [0,1] is less than or equal to the probability of mutation p_m, go to step 3. Otherwise, go to step 4.

Step 3. If another real random number rand() in [0,1] is less than or equal to a constant *R*, determine $\mu_{s(j)}$ randomly according to the Gaussian distribution with mean $\hat{\mu}_{s(j)}$ and variance τ^2, and go to step 4. Otherwise, determine $\mu_{s(j)}$ randomly according to the uniform distribution in [0, v_j], and go to step 4.

Step 4. If $j < n$, let $j = j+1$ and return to step 2. Otherwise, go to step 5.

Step 4. If $w < N$, let $w = w+1$ and return to step 1. Otherwise, stop.

Inversion for double strings

Step 0. Let $w = 1$.

Step 1. Generate a real random number rand() in [0,1]. For a given crossover rate p_i, if rand()$\le p_i$, then go to step 2. Otherwise, go to step 4.

Step 2. Choose two points h, k $(h \ne k)$ from {1,2,…,*n*} at random. Then, set $l = h$.

Step 3. Let $j = ((l-1)\%n)+1$. Then, interchange $(s(j), g_{s(j)})^T$ with $(s((n+k-(l-h)-1)\%n+1),\ \ g_{s((n+k-(l-h)-1)\%n+1)})^T$. Furthermore, set $l = l+1$ and go to step 4.

Step 4. 1) If $h < k$ and $l < h + \lfloor (k-h+1)/2 \rfloor$, return to step 3. If $h < k$ and $l \geq h + \lfloor (k-h+1)/2 \rfloor$, go to step 5. 2) If $h > k$ and $l < h + \lfloor (k+n-h+1)/2 \rfloor$, return to step 3. If $h > k$ and $l \geq h + \lfloor (k+n-h+1)/2 \rfloor$, go to step 5.

Step 5. If $w < N$, let $w = w+1$ and return to step 1. Otherwise, stop.

Observe that inversion is not only between *h* and *k* but also between *k* and *h*.

5.5. GADSCRRSU algorithm

Step 0. Set the values of parameters used in GADSCRRSU: the population size *N*, the generation gap *G*, the probability of crossover p_c, the probability of mutation p_m, the probability of inversion p_i, the minimal search generation $I_{\min}$, the maximal search generation $I_{\max} > I_{\min}$, the degree of using the information about solutions to continuous programming relaxation problems *R*. Set generation counter *t*=0.

Step 1. Generate the initial population consisting of *N* individuals based on the information of a solution to the continuous relaxation problem (22).

Step 2. Solve the system of differential equations in (21) and compute $P_1(t)$ for each individual and then decode each individual (genotype) in the current population and calculate its fitness based on the corresponding solution (phenotype).

Step 3. If the termination condition is fulfilled, then go to step 8. Otherwise, set *t*= *t*+1 and then go to step 4.

Step 4. Apply the reproduction operator based on the elitist expected value selection, after carrying out linear scaling.

Step 5. Apply the crossover operator, called PMX (Partially Matched Crossover) for double strings.

Step 6. Apply the mutation operator based on the information of an optimal solution to the continuous relaxation problem (22).
Step 7. Apply the inversion operator. Go to step 2.
Step 8. Stop.

6. Computational experiments

To demonstrate the efficiency of the proposed genetic algorithm method (GADSCRRSU), we solve two typical small and medium cases with different configurations. The objective is to obtain the optimal capacities using the GA. All experiments are replicated four times using different random number seeds.

6.1. Case I

The first case, which is depicted in Figure 3, has been taken from Azaron *et al.* [1]. This system produces chairs. The final chair consists of two separate parts: wooden and leather. In each node, except node 4, there is a manufacturing station with one machine. Node 4 contains an assembly station with one machine. The set of allowable capacity choices is considered to be S_i={11,11.5,12,12.5,…,19.5,20}, for all service rates.

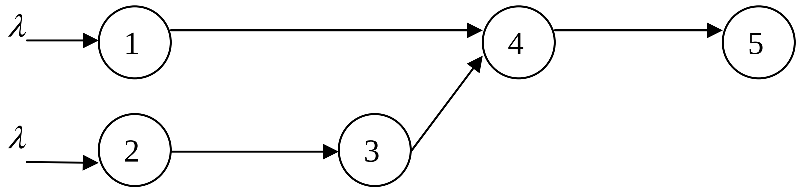

Figure 3. Dynamic assembly system of Case I

Table 1 shows the characteristics of the service stations in this case (cost unit is in dollar and time unit is in day). The given threshold *u* is equal to *3* days. The other assumptions are as follows:

1. The demand rate λ is equal to *10* per day.
2. The transport times between the service stations settled in nodes 1 and 4, and also between those settled in nodes 3 and 4 are independent exponentially distributed random variables with the parameters $\lambda_{(1,4)} = 1$ and $\lambda_{(3,4)} = 2$. The transport times between the other service stations are zero.

Table 1. Characteristics of the service stations in Case I

Service station	1	2	3	4	5
$C_i(\mu_i)$	$10\mu_1 + 4$	$4\mu_2 + 3$	$5\mu_3 + 7$	$\mu_4^2 + 2$	$2\mu_5 + 5$

The stochastic process *{X(t),t ≥ 0}* related to the longest path analysis of the corresponding transformed stochastic network has 14 states. We set the goals as $b_1=400$, $b_2=1.5$, $b_3=0.5$ and $b_4=0.9$. We solve the problem for the following sets of *c* to generate a set of Pareto-optimal solutions, according to the goal attainment formulation (18).

Set 1: ($c_1=0.5556$, $c_2=0.0556$, $c_3=0.1111$, $c_4=0.2777$),
Set 2: ($c_1=0.7407$, $c_2=0.037$, $c_3=0.037$, $c_4=0.1853$),
Set 3: ($c_1=0.8196$, $c_2=0.0164$, $c_3=0.082$, $c_4=0.082$),
Set 4: ($c_1=0.9615$, $c_2=0.0096$, $c_3=0.0096$, $c_4=0.0193$).

For example, according to the second set, one day deviation from the average lead time is considered to be as important as its variance and also 20 and 5 times as important as one dollar deviation from the total operating costs, and the probability that the manufacturing lead time does not exceed *3* days, respectively. ε is considered to be equal *0.05* in both cases.

Now, the proposed genetic algorithm (GADSCRRSU) is applied to solve the problem. We set the values of the parameters as: $N=25$, $G=0.9$, $p_c=0.9$, $p_m=0.05$, $p_i=0.03$, $I_{\min}=100$, $I_{\max}=500$, $R=0.8$ and $t=0$, refer to Sakawa [19] for the details about setting these parameters. Finally, the obtained Pareto-optimal solutions of Case I, according to 4 indicated sets of *c*, including the computational times, on a PC Pentium IV 2.1 GHz Processor, and the optimal resources are all given in Table 2.

Table 2. Pareto-optimal solutions of Case I

Set	z	$f_1(\mu)$	$f_2(\mu)$	$f_3(\mu)$	$f_4(\mu)$	Computational time (sec.)	μ_1^*	μ_2^*	μ_3^*	μ_4^*	μ_5^*
1	*25.972*	*414*	*2.944*	*2.005*	*0.594*	*15.7*	*12*	*13.5*	*13*	*11*	*16.5*
2	*31.389*	*423.25*	*2.647*	*1.501*	*0.683*	*15.34*	*12*	*13.5*	*13*	*11.5*	*15.5*
3	*55.621*	*444.75*	*2.412*	*1.385*	*0.75*	*15.96*	*13*	*14.5*	*13.5*	*11.5*	*18*
4	*69.423*	*466.75*	*2.161*	*1.103*	*0.826*	*15.37*	*13*	*14*	*13.5*	*12.5*	*18*

6.2. Case II

Case II, which is a medium scale case, is depicted in Figure 4. This system produces winter jackets. The final jacket consists of three parts: nylon, polyester and feather. There is one machine in the service stations settled in nodes 1, 2, 3 and 4, and infinite number of machines in the service stations settled in the other nodes. Table 3 shows the characteristics of the service stations in this case. The other assumptions are as follows:

1. $\lambda = 6$.
2. The transport time between the service stations settled in nodes 1 and 7 has generalized Erlang distribution of order 2 with the parameters $(\lambda_{(1,7)_1}, \lambda_{(1,7)_2}) = (3,5)$. The transport times between the other service stations are equal zero.

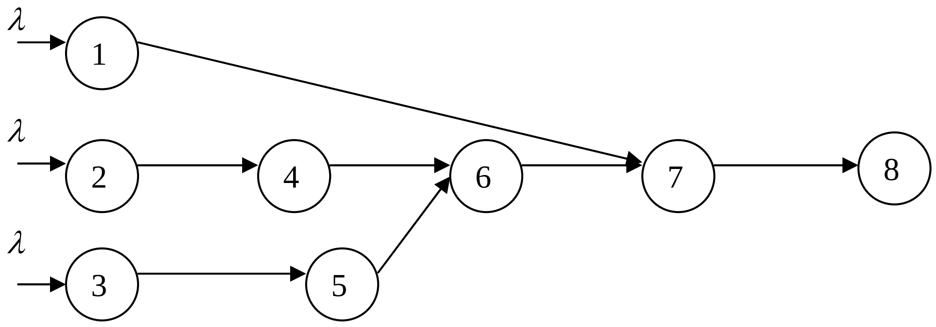

Figure 4. Dynamic assembly system of Case 2

Table 3. Characteristics of the service stations in Case II

Service station	1	2	3	4	5	6	7	8
$C_i(\mu_i)$	μ_1^2+3	μ_2	$2\mu_3$	$5\mu_4$	μ_5+1	$3\mu_6$	μ_7+2	$2\mu_8+4$

The corresponding stochastic process *$\{X(t), t \geq 0\}$* has 42 states. The given threshold is equal to *2.5* days in this case. It is also assumed that S_i={4,4.2,4.4,4.6,…,7.8,8} for *$i=1,2,\ldots,8$*. We set the goals as *$b_1=135$, $b_2=1.5$, $b_3=0.4$* and *$b_4=0.95$*. We also consider the same sets of *c* as the first case. The Pareto-optimal solutions and the optimal capacities of Case II are given in Tables 4 and 5, respectively.

Table 4. Pareto-optimal solutions of Case II

Set	z	$f_1(\mu)$	$f_2(\mu)$	$f_3(\mu)$	$f_4(\mu)$	Computational time (sec.)
1	*10.031*	*140.56*	*2.058*	*0.655*	*0.758*	*73.99*
2	*12.443*	*144.16*	*1.96*	*0.612*	*0.797*	*74.37*
3	*19.57*	*151.04*	*1.82*	*0.477*	*0.851*	*72.15*
4	*23.006*	*156.84*	*1.721*	*0.434*	*0.884*	*73.43*

Table 5. Optimal capacities of Case II

Set	μ_1^*	μ_2^*	μ_3^*	μ_4^*	μ_5^*	μ_6^*	μ_7^*	μ_8^*
1	*6.4*	*7.6*	*7.8*	*7*	*4*	*4.2*	*5.2*	*4.8*
2	*6.4*	*7.8*	*7.6*	*7.4*	*4.4*	*4.2*	*6.6*	*4.8*
3	*6.8*	*7.8*	*8*	*7.2*	*4.2*	*4.4*	*6.8*	*5.4*
4	*6.8*	*8*	*7.6*	*7.8*	*5.2*	*4.2*	*7.4*	*6.6*

As we explained in Section 4, solving the goal attainment formulation (18), optimally, and consequently, comparing the genetic algorithm results against the optimal results is impossible. Therefore, we try to compare the genetic algorithm results against the results of a discrete-time approximation of the formulation (18), proposed by Azaron *et al.* [1].

6.3. Comparison the GA results against those of discrete-time approximation technique

We use LINGO 6, in the same computer, to solve Cases I and II for 4 indicated sets of *c*, considering the discrete-time approximation technique. In this technique, we discretize the continuous-time system and convert the optimal control problem into an equivalent nonlinear programming problem. In other words, we transform the differential equations into equivalent difference equations as well as transform the integral terms into equivalent summation terms. To follow this approach, the time interval is divided into K equal portions with length Δt, refer to [1] for details. In each comparison, two different levels of K and Δt ($K=20, \Delta t=0.3$) and ($K=200, \Delta t=0.03$) are considered for both cases.

Figure 5 shows the objective function values *z* for the two indicated methods. According to this figure, the objective function values in both cases, using the genetic algorithm, are much less than those, considering the discrete-time approximation with *K=20*.

The objective function values, using the discrete-time approximation with *K=200*, are optimal or very near to optimal. Therefore, the percentage differences between the objective function values using the GA and the discrete-time approximation with *K=200*, or the absolute differences between *z* obtained from the GA and *z* obtained from the discrete-time approximation with *K=200* divided by *z* obtained from the GA, can be considered as a measure for assessing how good the GA results actually are. The maximum percentage differences in Cases I and II are equal *7.75%* and *3.12%*, respectively. Clearly, the GA results are very near to optimal results. Therefore, the efficiency of the proposed genetic algorithm approach is concluded.

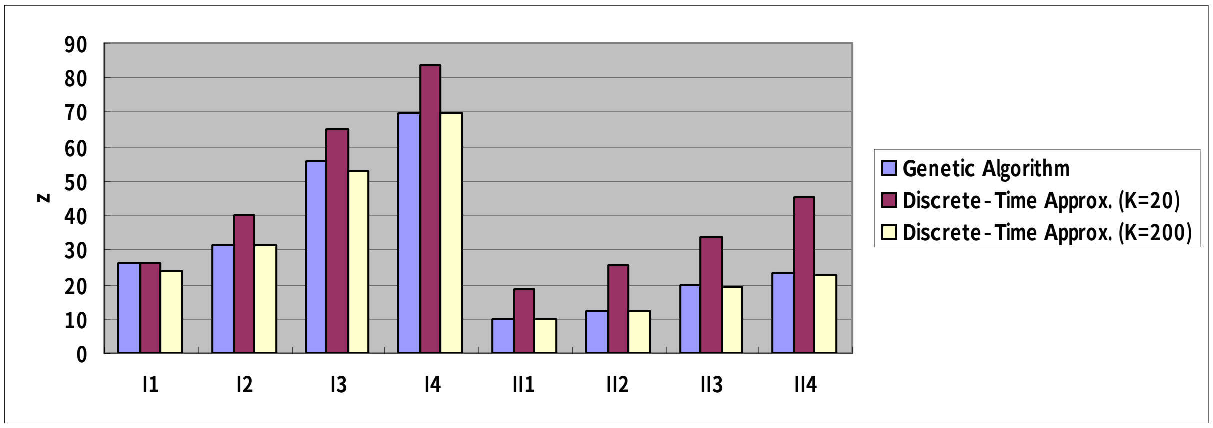

Figure 5. Objective function values (*z*) for Cases I and II, according to sets 1,2,3,4 of *c*

Figure 6 shows the computational times for the genetic algorithm and the discrete-time approximation, respectively. According to this figure, the computational times, using the genetic algorithm, are remarkably decreased, even comparing against the discrete-time approximation with *K=20*, especially for larger-scale cases. For example, the maximum computational time in the two indicated cases, using the GA, is equal to *74* seconds, but the required computational time to solve set 4 of Case II, using the discrete-time approximation with *K=200*, is about *14* hours. Therefore, it is clearly concluded that the genetic algorithm approach is computationally superior in terms of finding optimal or near-optimal solutions to large-scale problems than the discrete-time approximation technique.

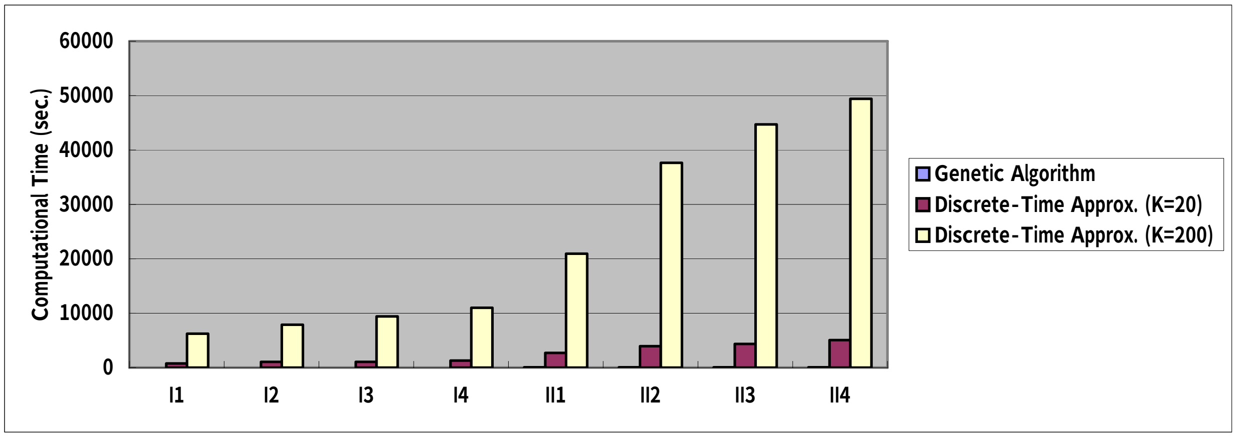

Figure 6. Computational times (sec.) for Cases I and II, according to sets 1,2,3,4 of *c*

7. Conclusion

In this paper, we introduced a genetic algorithm approach to control the service rates of the manufacturing and the assembly operations in a dynamic multistage assembly system, in which the average lead time, the variance of the lead time and the total operating costs of the system per period are minimized and the probability that the manufacturing lead time does not exceed a certain threshold is maximized.

To solve the relevant multi-objective programming and generating the Pareto-optimal solutions, we used the goal attainment method, which is a variation of the goal programming technique. Goal attainment method is one of the multi-objective techniques with priori articulation of preference information given. The goal attainment method has fewer variables to work with, so it will be computationally faster, and therefore is a good method to solve our problem.

The problem considered in this paper has discrete decision variables and involves nonlinearity. After the reformulation of the problem, we proposed a genetic algorithm with double strings using continuous relaxation based on reference solution updating (GADSCRRSU) to solve the problem.

According to the numerical experiments of Section 6, it is seen that the genetic algorithm method is an efficient method for the multi-objective lead time control problem.

We could also obtain the distribution function of the manufacturing lead time. Seidmann and Smith [21] have developed procedures to assign due-dates for jobs in a job shop environment assuming that the probability distribution of the lead time is known. Therefore, our results complement theirs. Together one may now assign due dates for the final product in a multistage assembly system.

Monte Carlo simulation can also be used to analyze the impact of non-exponential inter-arrival, processing and transport times.

We just obtain the lead time distribution in multistage assembly systems by transforming the related queueing network into an equivalent stochastic network and then computing the longest path distribution in the stochastic network by constructing a proper continuous-time Markov chain. Any other analytical longest path approach could also be used to obtain the lead time distribution.

The contribution of this paper is to provide a framework to deal with the optimal service control in multistage assembly systems with infinite buffer capacities, considering the role of transport times between service stations, using genetic algorithms.

In this paper, we only consider policies that are fixed at time zero. Such policies can be sub-optimal as they can be bettered by dynamic policies. May be in the future, this paper can be extended to obtain optimal service control, dynamically, in multistage assembly systems using MDPs, refer to Bertsekas [3].

References

[1] Azaron, A., Katagiri, H., Kato, K., Sakawa, M., 2005. Modelling Complex Assemblies as a Queueing Network for Lead Time Control. To appear in European Journal of Operational Research.

[2] Azaron, A., Modarres, M., 2005. Distribution Function of the Shortest Path in Networks of Queues. OR Spectrum 27, 123-144.

[3] Bertsekas, D.P., 2001. Dynamic Programming and Optimal Control, Vol. 2. Athena Scientific, MA.

[4] Cheng, T.C.E., Gupta, M.C., 1989. Survey of Scheduling Research Involving Due Date Determination decisions. European Journal of Operational Research 38, 156-166.

[5] Gold, H., 1998. A Markovian Single Server with Upstream Job and Downstream Demand Arrival Stream. Queueing Systems 30, 435-455.

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

[7] Goldberg, D.E., Lingle, R., 1985. Alleles, Loci, and the Traveling Salesman Problem. Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications. Lawrence Erlbaum Associates, Hillsdale, NJ, 154-159.

[8] Harrison, J.M., 1973. Assembly-Like Queues. Journal of Applied Probability 10, 354-367.

[9] Haskose, A., Kingsman, B.G., Worthington, D., 2002. Modelling Flow and Jobbing Shops as a Queueing Network for Workload Control. International Journal of Production Economics 78, 271-285.

[10] Hemachandra, N., Eedupuganti, S.K., 2003. Performance Analysis and Buffer Allocations in Some Open Assembly Systems. Computers and Operations Research 30, 695-704.

[11] Kapadia, A.S., Hsi, B.P., 1978. Steady State Waiting Time in a Multicenter Job Shop. Naval Research Logistics Quarterly 25, 149-154.

[12] Kulkarni, V., Adlakha, V., 1986. Markov and Markov-Regenerative PERT Networks. Operations Research 34, 769-781.

[13] Lemoine, A.J., 1979. On Total Sojourn Time in Networks of Queues. Management Science 25, 1034-1035.

[14] Lipper, E.H., Sengupta, B., 1986. Assembly-Like Queues with Finite Capacity: Bounds, Asympototics and Approximations. Queueing Systems 1, 67-83.

[15] Michalewicz, Z., 1996. Genetic algorithms + Data Structures = Evolution Programs. 3rd revised and extended edition, Springer-Verlag, Berlin.

[16] Papadopoulos, H.T., Heavey, C., 1996. Queueing Theory in Manufacturing Systems Analysis and Design: A Classification of Models for Production and Transfer Lines. European Journal of Operational Research 92, 1-27.

[17] Ramachandran, S., Delen, D., 2005. Performance Analysis of a Kitting Process in Stochastic Assembly Systems. Computers and Operations Research 32, 449-463.

[18] Sakawa, M., Kato, K., 2003. Genetic Algorithms with Double Strings for 0-1 Programming Problems. European Journal of Operational Research 144, 581-597.

[19] Sakawa, M., 2001. Genetic Algorithms and Fuzzy Multiobjective Optimization. Kluwer Academic Publishers, Boston.

[20] Sakawa, M., Kato, K., Sunada, H., Shibano, T., 1997. Fuzzy Programming for Multiobjective 0-1 Programming Problems through Revised Genetic Algorithms. Eurpean Journal of Operational Research 97, 149-158.

[21] Seidmann, A., Smith, M.L., 1981. Due Date Assignment for Production Systems. Management Science 27, 571-581.

[22] Sethi, S., Thompson, G., 1981. Optimal Control Theory. Martinus Nijhoff Publishing, Boston.

[23] Song, D.P., Hicks, C., Earl, C.F., 2002. Product Due Date Assignment for Complex Assemblies. International Journal of Production Economics 76, 243-256.

[24] Vandaele, N., Boeck, L.D., Callewier, D., 2002. An Open Queueing Network for Lead Time Analysis. IIE Transactions 34, 1-9.

[25] Yano, C.A., 1987. Stochastic Lead-Time in Two-Level Assembly Systems. IIE Transactions 19, 371-378.