Available online at www.sciencedirect.com
SCIENCE DIRECT®
Applied Soft Computing 6 (2005) 38–52
Applied Soft Computing
www.elsevier.com/locate/asoc

New hybrid genetic operators for real coded genetic algorithm to compute optimal control of a class of hybrid systems

M. Senthil Arumugam[a,*], M.V.C. Rao[a], Ramaswamy Palaniappan[b]

[a]*Faculty of Engineering and Technology, Multimedia University, Jalan Ayer Keroh, 75450 Malacca, Malaysia*
[b]*Biomedical Engineering Research Centre, Nanyang Technological University, 637553 Singapore, Singapore*

Received 13 April 2004; received in revised form 13 October 2004; accepted 11 November 2004

Abstract

This paper introduces new hybrid cross-over methods and new hybrid selection methods for real coded genetic algorithm (RCGA), to solve the optimal control problem of a class of hybrid system, which is motivated by the structure of manufacturing environments that integrate process and optimal control. In this framework, the discrete entities have a state characterized by a temporal component whose evolution is described by event-driven dynamics and a physical component whose evolution is described by continuous time-driven systems. The proposed RCGA with hybrid genetic operators can outperform the conventional RCGA and the existing Forward Algorithms for this class of systems. The hybrid genetic operators improve both the quality of the solution and the actual optimum value of the objective function. A typical numerical example of the optimal control problem with the number of jobs varying from 5 to 25 is included to illustrate the efficacy of the proposed algorithm. Several statistical analyses are done to compare the betterment of the proposed algorithm over the conventional RCGA and Forward Algorithm. Hypothesis *t*-test and Analysis of Variance (ANOVA) test are also carried out to validate the effectiveness of the proposed algorithm.

Keywords: Hybrid systems; Optimal control; Genetic algorithm (GA); Real coded genetic algorithm (RCGA); Hybrid genetic operators

1. Introduction

Hybrid systems are characterized by the combination of time-driven and event-driven dynamics in which the former are represented by differential (or difference) equations and the latter by discrete event systems. Two categories of modelling framework have been proposed to study hybrid systems. Those that extend event-driven models to include time-driven dynamics; those that extend the traditional time-driven models to include event-driven dynamics [1–3].

The hybrid system-modeling framework considered in this research falls into the first category. It is motivated by the structure of many manufacturing systems. In these systems, discrete entities (referred to as *jobs*) move through a network of work centers,

* Corresponding author.
E-mail address: msenthil.arumugam@mmu.edu.my (M.S. Arumugam).

1568-4946/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2004.11.001

which process the jobs so as to change their physical characteristics according to certain specifications. Each job is associated with a *temporal state* and a *physical state*. The temporal state of a job evolves according to event-driven dynamics and includes information such as the processing time or departure time of the job. The physical state evolves according to the time-driven dynamics and describes some measures of ''quality'' of the job such as temperature, weight, and chemical composition. The interaction of time-driven with event-driven dynamics leads to a natural tradeoff between temporal requirements on job completion times and physical requirements on the quality of the completed jobs. Such modelling frameworks and optimal control problems have been considered in [1–3].

By the nature of the event-driven dynamics, the problem is inherently nonconvex and nondifferentiable. Moreover, its dimension (number of independent variables) is identical to the number of considered jobs. If the number of variables is in the hundreds or thousands, the problem is highly complex and defies general-purpose algorithms like dynamic programming, backward recursive algorithms [4], and Forward Algorithms [5] for its solution. The task of solving these problems was simplified by exploiting structural properties of the optimal sample path. In particular, an optimal sample path is decomposed into decoupled segments, termed *busy periods*. Moreover each busy period is further decomposed into *blocks* defining certain jobs termed *critical jobs*. Identifying such critical jobs and their properties was a crucial part of the analysis and the key to developing effective algorithms for solving the optimal control problems. The identification of critical jobs and busy periods has been realized using nonsmooth optimization techniques [2].

Three algorithms were developed for solving such problems. They decompose the entire optimal control problem into a set of smaller convex optimization sub problems with linear constraints. The first is a ''backward'' recursive algorithm [4] proceeding backward in time from the last job to first job. The complexity of the problem was thus reduced from exponential N (the number of jobs processed) to a linearly bounded one by $2N - 1$. The second is a ''forward'' algorithm whose complexity is simply N as shown in [5]. The third is an ''improved forward'' algorithm, which is the extension of forward algorithm. Instead of increasing the number of jobs by one at every step, this algorithm may increase the number of jobs by more than one [2].

In this paper, we propose a real coded genetic algorithm (RCGA) with different forms of selection methods and cross-over methods. The selection methods adopted in this paper are Roulette wheel selection (RWS), Tournament selection (TS) and the hybrid combination of the both. The cross-over operation is carried out with combination of three different methods: Arithmetic cross-over (AMXO), Average Convex cross-over (ACXO), and Direction based cross-over (DBXO). We use Dynamic Mutation (DM) as a third genetic operator.

This paper is divided into six sections. In Section 2, the optimal control problem of a single stage hybrid manufacturing system is studied and formulated. Section 3 gives an overview of RCGA and provides the justification of the choice of RCGA for solving the optimal control problem. The design of hybrid genetic operators for various RCGA methods is discussed in Section 4. Section 5 depicts the numerical examples, the simulation results and the statistical analyses of various RCGA methods and finally the discussions and conclusions are presented in Section 6.

2. Problem formulation of single stage hybrid manufacturing system

The single stage hybrid system framework with event-driven and time-driven dynamics is illustrated in Fig. 1.

A sequence of N jobs is assigned by an external source to arrive for processing at known times $0 \leq a_1 \leq \ldots \leq a_N$. The jobs are denoted by C_i, $i = 1, 2, \ldots, N$. The jobs are processed first-come first-serve (FCFS) basis by a work-conserving and nonpreemptive server. The processing time is $s(u_i)$, which is a function of a control variable u_i, and $s(u_i) \geq 0$.

2.1. Time-driven dynamics

The time-driven dynamics is defined by the equation which evolved the job C_i which is initially at some physical state ξ_i at time x_0.

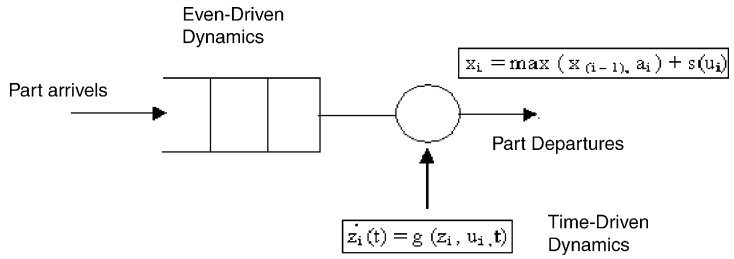

Fig. 1. A single stage hybrid manufacturing system.

$$\dot{z}_i(t) = g(z_i, u_i, t), \qquad z_i(x_0) = \xi_i \tag{1}$$

Note that Eq. (1) assumes that the jobs are noninteracting in that the physical state and controls of other jobs do not effect the dynamics of the ith job.

2.2. Event-driven dynamics

The event-driven dynamics is described by recursive nonlinear equations involving a maximum or a minimum operation, which is typically found in models of discrete event systems (DES). For the fist-come first-serve (FCFS), nonpreemptive, single server example in Fig. 1, these dynamics is given by the standard Lindley equation of the form:

$$x_i = \max(x_{(i-1)}, a_i) + s(u_i), \qquad i = 1, \ldots, N \tag{2}$$

where x_i is the departure or completion time of ith job.

2.3. Control objective function

Note that the choice of control u_i affects both the physical state z_i and next temporal state x_i, and thus time-driven dynamics (1) and event-driven dynamics (2), justifying the hybrid nature of the system [1]. According to [1], there are two alternative ways to view this hybrid system. The first is as a discrete event queuing system with time-driven dynamics evolving during processing in the server, as shown in Fig. 2.

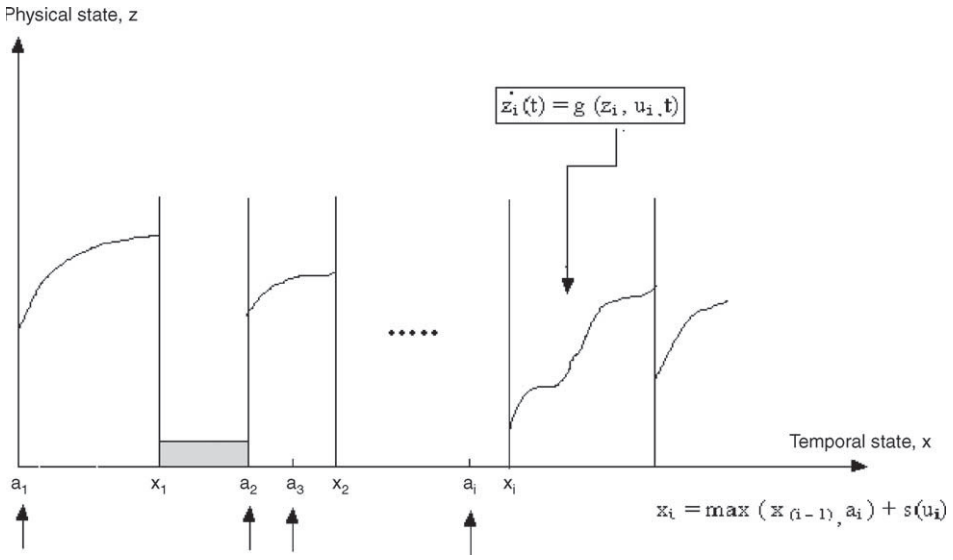

Fig. 2. Typical trajectory.

The other viewpoint interprets the model as a switched system. In this framework, each job must be processed until it reaches a certain quality level denoted by Γ_i. That is, the processing time for each job is chosen such that

$$s_i(u_i) = \min\left[t \geq 0;\ z_i(t_0) = \int_{t_0}^{t_0+t} g_i(\tau, u_i, t)\,\mathrm{d}\tau + z(t_0) \in \Gamma_i\right] \quad (3)$$

Fig. 2 shows the evolution of the physical state as a function of time t. It is shown in the figure that the dynamics of the physical state experience a ''switch'' when certain events occur. These events may classify into two categories: uncontrolled (or exogenous) arrival events and controlled departure events. In Fig. 2, the first event is an exogenous arrival event at time a_1. When the first job arrives at a_1, the physical state starts to evaluate the time-driven dynamics until it reaches the departure time x_1. Note that the first job completes before the second job arrives and hence there is an idle period, in which the server has no jobs to process. The physical state again begins evolving the time-driven dynamics at time a_2 (arrival of second job) until the second job completes at x_2. Note, however, that the third job arrived before the second job was completed. So the third job is forced to wait in the queue until time x_2. After the second job completes at x_2 the physical state begins to process the third job. As indicated in Fig. 2, not only do the arrival time and departure time cause switching in the time-driven dynamics according to (1), but the sequence in which these events occur is governed by the event-driven dynamics given in (2).

For the above single stage framework defined by Eqs. (1) and (2), the optimal control objective is to choose a control sequence $\{u_1, \ldots, u_N\}$ to minimize an objective function of the form:

$$J = \sum_{i=1}^{N} \{\theta_i(u_i) + \phi_i(x_i)\} \quad (4)$$

Although, in general, the state variables $z_i, \ldots, z_N$ evolve continuously with time, minimizing (4) is an optimization problem in which the values of the state variables are considered only at the job completion times $x_1, \ldots, x_N$. Since the stopping criterion in (3) is used to obtain the service times, a cost on the physical state $z_i(x_i)$ is unnecessary because the physical state of each completed job satisfies the quality objectives, i.e., $z_i(x_i) \in \Gamma_i$.

Generally speaking, u_i is a control variable affecting the processing time through $s_i = s(u_i)$ for extensions to cases with time-varying controls $u_i(t)$ over a service time. By assuming $s_i(\cdot)$ is either monotone increasing or monotone decreasing, given a control u_i, service time s_i can be determined from $s_i = s(u_i)$ and vice versa.

Table 1
16 RCGA method of the proposed algorithm

Name of the method	Selection	Cross-over	Mutation
RCGA_a1	TS	AMXO + ACXO	DM
RCGA_a2	TS	AMXO + DBOXO	DM
RCGA_a3	TS	ACXO + DBOXO	DM
RCGA_a4	TS	AMXO + DBOXO + ACXO	DM
RCGA_b1	RWS	AMXO + ACXO	DM
RCGA_b2	RWS	AMXO + DBOXO	DM
RCGA_b3	RWS	ACXO + DBOXO	DM
RCGA_b4	RWS	AMXO + DBOXO + ACXO	DM
RCGA_c1	RWS + TS	AMXO + ACXO	DM
RCGA_c2	RWS + TS	AMXO + DBOXO	DM
RCGA_c3	RWS + TS	ACXO + DBOXO	DM
RCGA_c4	RWS + TS	AMXO + DBOXO + ACXO	DM
RCGA_d1	RWS + TS + RWS + TS	AMXO + ACXO	DM
RCGA_d2	RWS + TS + RWS + TS	AMXO + DBOXO	DM
RCGA_d3	RWS + TS + RWS + TS	ACXO + DBOXO	DM
RCGA_d4	RWS + TS + RWS + TS	AMXO + DBOXO + ACXO	DM

For simplicity, let $s_i = u_i$ and the rest of the analysis is carried out with the notation u_i. Hence the optimal control problem, denoted by P is of the following form:

$$P: \min_{u_1,\ldots,u_N} \left\{ J = \sum_{i=1}^{N} \{\theta_i(u_i) + \phi_i(x_i)\} \quad : u \geq 0, \; i = 1,\ldots,N \right\} \tag{5}$$

subject to

$$x_i = \max(x_{(i-1)}, a_i) + s(u_i), \quad i = 1,\ldots,N \tag{6}$$

The optimal solution of P is denoted by u_i^* for$i = 1,\ldots,N$, and the corresponding departure time in Eq. (6) is denoted by x_i^* for$i = 1,\ldots,N$.

A common manufacturing problem is to produce jobs that meet certain minimum quality standards and deliver them by specified due dates. To achieve this, a cost on missing due dates are placed with the job completion times.Let us consider

$$\theta(u_i) = \frac{1}{u_i} \quad \text{and} \quad \phi_i(x_i) = (x_i - d_i)^2 \tag{7}$$

Letting d_i be the due date of job i, the cost on the departure time penalizes job earliness and tardiness. For a specific manufacturing application of this frame-

Table 2
The analysis of the fitness values for $N = 5$

Method	Average	S.D.	CV	AVEDEV
RCGA_a1	34.82655	0.00484	0.00014	0.00421
RCGA_a2	34.83352	0.00894	0.00026	0.00739
RCGA_a3	34.83330	0.00693	0.00020	0.00553
RCGA_a4	34.82887	0.00524	0.00015	0.00493
RCGA_b1	34.87133	0.01946	0.00056	0.01562
RCGA_b2	34.89751	0.04411	0.00126	0.03175
RCGA_b3	34.89400	0.03768	0.00108	0.03230
RCGA_b4	34.88014	0.03438	0.00099	0.02004
RCGA_c1	34.81719	0.00065	0.00002	0.00055
RCGA_c2	34.81820	0.00082	0.00002	0.00064
RCGA_c3	34.81806	0.00083	0.00002	0.00072
RCGA_c4	34.81751	0.00081	0.00002	0.00063
RCGA_d1	34.81759	0.00067	0.00002	0.00051
RCGA_d2	34.82120	0.00420	0.00012	0.00365
RCGA_d3	34.81925	0.00172	0.00005	0.00146
RCGA_d4	34.81763	0.00071	0.00002	0.00055

Methods	P_Value	Best method
1 and 2	0.000199	1
1 and 3	0.000026	1
1 and 4	0.068918	1
1 and 5	0.000000	1
1 and 6	0.000000	1
1 and 7	0.000000	1
1 and 8	0.000000	1
1 and 9	1.000000	9
9 and 10	0.000001	9
9 and 11	0.000020	9
9 and 12	0.042935	9
9 and 13	0.009741	9
9 and 14	0.000002	9
9 and 15	0.000000	9
9 and 16	0.007401	9

Table 3
The analysis of the fitness values for $N = 10$

Method	Average	S.D.	CV	AVEDEV
RCGA_a1	102.7144	0.155111	0.00151	0.117572
RCGA_a2	102.8805	0.155966	0.001516	0.118182
RCGA_a3	102.7316	0.202768	0.001974	0.174509
RCGA_a4	102.7241	0.090966	0.000886	0.069843
RCGA_b1	102.8006	0.195844	0.001905	0.165192
RCGA_b2	102.9028	0.128983	0.001253	0.105035
RCGA_b3	102.8065	0.200122	0.001947	0.16944
RCGA_b4	102.8017	0.15615	0.001519	0.109037
RCGA_c1	102.208	0.02918	0.000285	0.025234
RCGA_c2	102.2726	0.055023	0.000538	0.04262
RCGA_c3	102.2543	0.045929	0.000449	0.04002
RCGA_c4	102.2225	0.029854	0.000292	0.024357
RCGA_d1	102.2232	0.0291	0.000285	0.024235
RCGA_d2	102.332	0.065104	0.000636	0.053347
RCGA_d3	102.2758	0.048654	0.000476	0.036808
RCGA_d4	102.2508	0.050109	0.00049	0.04079

Methods	P_Value	Best method
1 and 2	0.00006	1
1 and 3	0.35731	1
1 and 4	0.38520	1
1 and 5	0.03185	1
1 and 6	0.00000	1
1 and 7	0.02556	1
1 and 8	0.01694	1
1 and 9	1.00000	9
9 and 10	0.00000	9
9 and 11	0.00001	9
9 and 12	0.03187	9
9 and 13	0.02436	9
9 and 14	0.00000	9
9 and 15	0.00000	9
9 and 16	0.00008	9

work, the reader is referred to [6] where a steel heating/annealing process is modelled, analysed, and optimised using a setting similar to (5) and (6).

3. Real coded genetic algorithm

The genetic algorithm (GA) is a search technique based on the mechanics of natural genetics and survival of the fittest. GA is attractive and alternative tool for solving complex multi-modal optimization problems [7,8]. GA is unique in that they operate from a rich database of many points simultaneously. Any carefully designed GA is only able to balance the exploration of the search effort, which means that an increase in the accuracy of a solution can only come at the sacrifice of convergent speed, and vice versa. It is unlikely that both of them can be improved simultaneously. Despite their superior search ability, GA still fails to meet the high expectations that theory predicts for the quality and efficiency of the solution. As widely accepted that a conventional GA (CGA) is only capable of identifying the high performance region at an affordable time and displays inherent difficulties in performing local search for numerical applications [7–10].

To improve the final local tuning capabilities of a binary coded genetic algorithm, which is a must for

Table 4
The analysis of the fitness values for N = 15

Method	Average	S.D.	CV	AVEDEV
RCGA_a1	214.5437	0.170857	0.000796	0.149185
RCGA_a2	214.8613	0.328986	0.001531	0.276374
RCGA_a3	214.6921	0.18379	0.000856	0.159528
RCGA_a4	214.5493	0.197774	0.000922	0.164395
RCGA_b1	215.5454	0.247873	0.00115	0.242789
RCGA_b2	215.802	0.225015	0.001043	0.197281
RCGA_b3	215.8427	0.198742	0.000921	0.164779
RCGA_b4	215.5931	0.251386	0.001166	0.214102
RCGA_c1	214.1902	0.098628	0.00046	0.06818
RCGA_c2	214.3163	0.092914	0.000434	0.07696
RCGA_c3	214.2689	0.051595	0.000241	0.046307
RCGA_c4	214.1933	0.039167	0.000183	0.029403
RCGA_d1	214.2235	0.054735	0.000256	0.043754
RCGA_d2	214.3295	0.074418	0.000347	0.065133
RCGA_d3	214.2829	0.082691	0.000386	0.069414
RCGA_d4	214.2254	0.056246	0.000263	0.0475

Methods	*P*_Value	Best method
1 and 2	0.00107	1
1 and 3	0.97369	3
3 and 4	0.00266	3
3 and 5	0.00000	3
3 and 6	0.00000	3
3 and 7	0.00000	3
3 and 8	0.00000	3
3 and 9	1.00000	9
9 and 10	0.00014	9
9 and 11	0.00000	9
9 and 12	0.43638	9
9 and 13	0.05566	9
9 and 14	0.00011	9
9 and 15	0.00000	9
9 and 16	0.04740	9

Table 5
The analysis of the fitness values for N = 20

Method	Average	S.D.	CV	AVEDEV
RCGA_a1	388.1877	0.206649	0.000532	0.172676
RCGA_a2	388.7952	0.23397	0.000602	0.184971
RCGA_a3	388.2683	0.233766	0.000602	0.180176
RCGA_a4	388.2283	0.072565	0.000187	0.16162
RCGA_b1	389.672	0.253681	0.000651	0.218732
RCGA_b2	389.8391	0.326392	0.000837	0.250175
RCGA_b3	389.7734	0.246922	0.000634	0.201272
RCGA_b4	389.7254	0.310312	0.000796	0.256618
RCGA_c1	387.5549	0.054139	0.00014	0.043558
RCGA_c2	387.8139	0.111055	0.000286	0.087905
RCGA_c3	387.78	0.101776	0.000262	0.078816
RCGA_c4	387.5573	0.055753	0.000144	0.040718
RCGA_d1	387.6347	0.072565	0.000187	0.06162
RCGA_d2	387.8998	0.075263	0.000194	0.066103
RCGA_d3	387.7627	0.126984	0.000327	0.097084
RCGA_d4	387.638	0.068648	0.000177	0.057513

Methods	*P*_Value	Best method
1 and 2	0.00000	1
1 and 3	0.08284	1
1 and 4	0.23937	1
1 and 5	0.00000	1
1 and 6	0.00000	1
1 and 7	0.00000	1
1 and 8	0.00000	1
1 and 9	1.00000	9
9 and 10	0.00000	9
9 and 11	0.00000	9
9 and 12	0.43304	9
9 and 13	0.00001	9
9 and 14	0.00000	9
9 and 15	0.00000	9
9 and 16	0.00000	9

high precision problems new genetic operators have been introduced [8,11]. The main objective behind real coded GA implementations is to move the genetic algorithm closer to the problem space. Such a move forces, but also allows, the operators to be more problem specific by utilizing some specific characteristics of real space.

For most applications of GAs to constrained optimization problems, the real coding is used to represent a solution to a given problem. Such coding is also known as floating-point representation, real number representation. In recent years, several genetic operators were also proposed for such coding [8,11], which can be roughly classified into: conventional operators, arithmetic operators and direction based operators.

Table 6
The analysis of the fitness values for $N = 25$

Method	Average	S.D.	CV	AVEDEV
RCGA_a1	638.0587	0.214725	0.000337	0.175276
RCGA_a2	638.6549	0.163925	0.000257	0.132333
RCGA_a3	638.1023	0.201296	0.000315	0.167882
RCGA_a4	638.0788	0.260332	0.000408	0.197322
RCGA_b1	640.4241	0.373136	0.000583	0.296146
RCGA_b2	640.5726	0.364959	0.00057	0.298320
RCGA_b3	640.6646	0.474439	0.000741	0.392971
RCGA_b4	640.6411	0.397675	0.000621	0.338996
RCGA_c1	637.3034	0.075930	0.000119	0.064133
RCGA_c2	637.6329	0.110003	0.000173	0.088947
RCGA_c3	637.5061	0.114456	0.00018	0.092771
RCGA_c4	637.3111	0.079096	0.000124	0.067401
RCGA_d1	637.4557	0.102425	0.000161	0.093902
RCGA_d2	637.6589	0.10882	0.000171	0.071385
RCGA_d3	637.6332	0.103342	0.000162	0.085240
RCGA_d4	637.4614	0.091697	0.000144	0.076326

Methods	*P*_Value	Best method
1 and 2	0.00000	1
1 and 3	0.21018	1
1 and 4	0.37216	1
1 and 5	0.00000	1
1 and 6	0.00000	1
1 and 7	0.00000	1
1 and 8	0.00000	1
1 and 9	1.00000	9
9 and 10	0.00000	9
9 and 11	0.00000	9
9 and 12	0.35089	9
9 and 13	0.00000	9
9 and 14	0.00000	9
9 and 15	0.00000	9
9 and 16	0.00000	9

GAs start searching the solution by initialising a population of random candidates to the solution. Every individual in the population undergoes genetic evolution through cross-over and mutation. The selection procedure is conducted based on the fitness of each individual. In this paper, the Roulette-wheel selection (RWS), Tournament selection (TS) and hybrid of both selection procedures are adopted in conjunction with the elitist strategy. By using elitist strategy, the best individual in each generation is ensured to be passed to the next generation.

The selection operator creates a new population (or generations) by selecting individuals from the old populations, biased towards the best. The chromosomes, which produce the best optimal fitness, are selected for next generations. Cross-over is the main genetic operator, consists of swapping chromosome parts between individuals. Cross-over is not performed on every pair of individuals; its frequency being controlled by a cross-over probability (P_c). The probability should have a larger value, typically, $P_c = 0.8$. The last operator is mutation and consists of changing a random part of string representing the individual. This operator must be used with some care, with low probability; typically P_m ranges from 0.01 to 0.1 for normal populations. GA operators are applied for several generations until one of the individuals of population converges to an optimal value or the required number of generations (max_gen) is reached.

The high level behavior of GA can be depicted as follows:

```
1. Start
2. Generate (OLDPOP)
3. Repeat until limit
   Evaluate (OLDPOP)
   NEWPOP = Select (OLDPOP)
   Cross-over (NEWPOP)
   Mutation (NEWPOP)
   OLDPOP = NEWPOP
4. End
```

Michaelewicz [8] indicates that for real valued numerical optimization problems, floating-point

representations outperform binary representations because they are more consistent, more precise, and lead to faster execution. For most applications of GAs to optimisation problems, the real coding technique is used to represent a solution to a given problem. Hence, the real coded genetic algorithm (RCGA) is considered in this paper for solving the optimal control problem.

4. Algorithm design

The main elements of a real coded GA include initial population, fitness function, genetic operators (selection, cross-over and mutation), genetic structure, parameters (max_gen, P_c, and P_m), etc. The following is the design of our proposed algorithm.

4.1. Initial population

The initial populations are generated randomly. And the number of chromosomes generated per population is equal to the dimension of the optimal problem or equal to the number of jobs (N) involved in the main objective function. In this paper the number of chromosomes generated per population (or the dimension of the optimal control problem) is varying from 5 to 25.

4.2. Selection

Three different types of selection methods are used in this paper: Roulette wheel method, Tournament selection method, and the hybrid combinations with different proportions of roulette wheel and tournament selection methods.

4.2.1. Roulette wheel selection method

Each individual in the population is assigned a space on the roulette wheel, which is proportional to the individual relative fitness. Individuals with the largest portion on the wheel have the greatest probability to be selected as parent generation for the next generation.

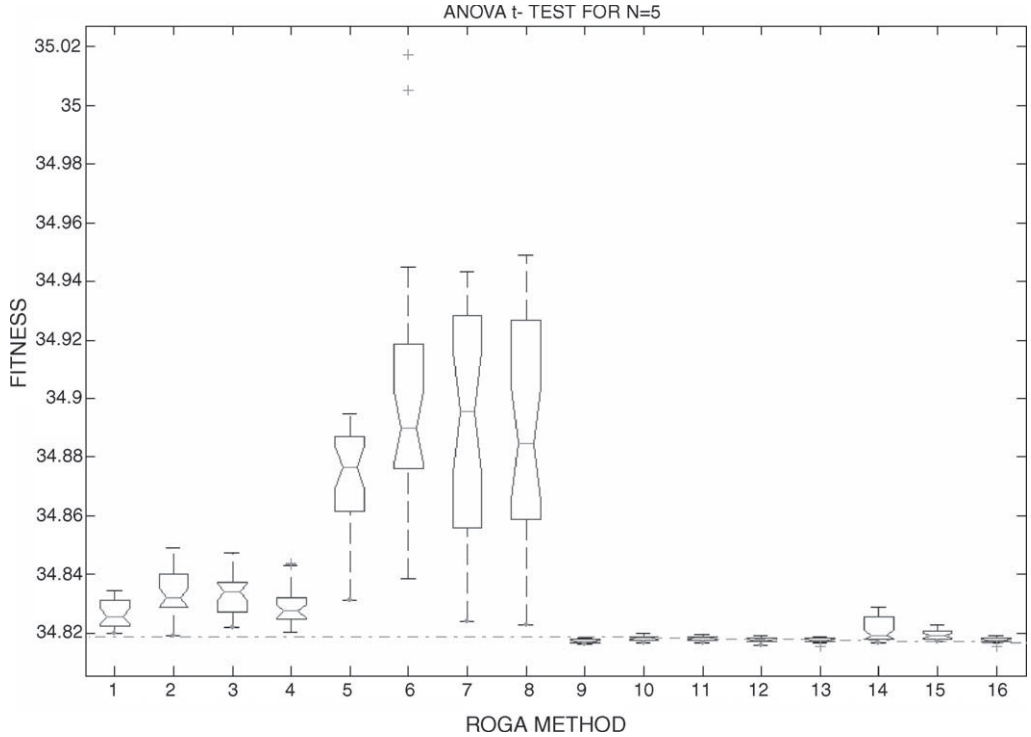

Fig. 3. ANOVA t-test for $N = 5$.

4.2.2. Tournament selection method

In tournament selection, a number *Tour* of individuals is chosen randomly from the population and the best individual from this group is selected as a parent. This process is repeated as often as individuals to choose. These selected parents produce uniform offspring at random. The parameter for tournament selection is the tournament size *Tour*. *Tour* takes values ranging from $2 - N_{ind}$ (number of individuals in population).

4.2.3. Hybrid selection method

The hybrid slection method consists of the combination of both RWS and TS. We designed two types of hybrid selections: single level and two level hybrid selection methods. In single level hybrid selection method, 50% of the population size adopts TS procedure where as the RWS procedure is used in the remaining 50% of the population size. In other words, the offspring is generated using these procedures, 50% using TS and another 50% using RWS. The two level hybrid selection method consists 25% of TS, then followed by 25% of RWS and again 25% of TS and 25% of RWS [12].

4.3. Cross-over

Cross-over is the main genetic operator and consists of swapping chromosome parts between individuals. Cross-over is not performed on every pair of individuals; its frequency being controlled by a cross-over probability (P_c). There are several cross-over methods available and here we use hybrid combination of Arithmetic cross-over method (AMXO), Average Convex cross-over (ACXO), and Direction based cross-over (DBXO).

4.3.1. Arithmetic cross-over method (AMXO)

The basic concept of this method is borrowed from the convex set theory [7,10]. Simple arithmetic operators are defined as the combination of two vectors (chromosomes) as follows:

$$x' = \lambda x + (1 - \lambda)y \tag{8}$$

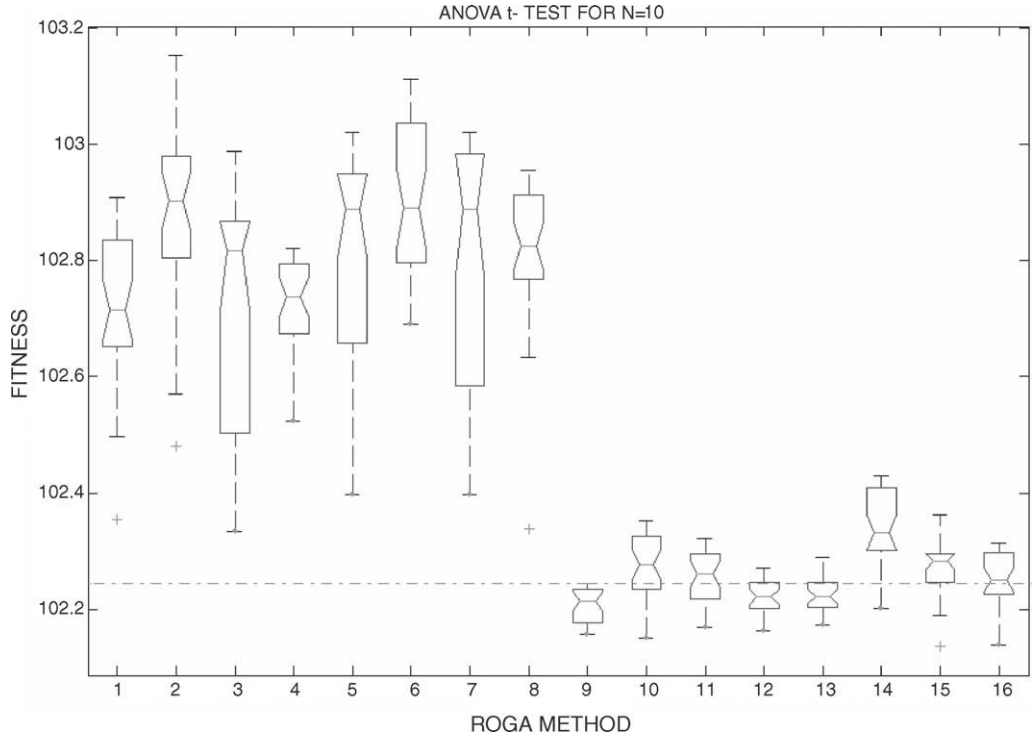

Fig. 4. ANOVA *t*-test for $N = 10$.

$$x'' = (1 - \lambda)x + \lambda y \tag{9}$$

where λ is a uniformly distributed random variable between 0 and 1.

4.3.2. Average convex cross-over method (ACXO)

Generally, the weighted average of two vectors x_1 and x_2 are calculated as follows:

$$\lambda_1 x_1 + \lambda_2 x_2 \tag{10}$$

if the multipliers are restricted as

$$\lambda_1 + \lambda_2 = 1, \quad \lambda_1 > 0, \ \lambda_2 > 0 \tag{11}$$

the weighted form (10) is known as *convex combination*.

Similarly, arithmetic operators are defined as the combination of two chromosomes as follows:

$$\begin{aligned} x_1' &= \lambda_1 x_1 + \lambda_2 x_2 \\ x_1' &= \lambda_1 x_2 + \lambda_2 x_1 \end{aligned} \tag{12}$$

According to the restriction of multipliers, it yields three kinds of cross-over, which can be called convex cross-over, affine cross-over, and linear cross-over. Among the three methods, convex cross-over may be the most commonly used method [7]. When restricting $\lambda_1 = \lambda_2 = 0.5$, it yields a special case, which is called as *Average convex cross-over* (ACXO) by Davis [10] or *intermediate cross-over* by Schwefel [11].

4.3.3. Direction-based cross-over (DBXO)

With the conventional genetic operators, there is no guarantee that the offspring are better than their parents. For the direction-based operators, problem-specific knowledge is introduced into genetic operators in order to produce improved offspring. Direction-based cross-over uses the values of the objective function in determining the direction of genetic search. The operator generates single offspring x' from two parents x_1 and x_2 according to the following rule:

$$x' = r(x_2 - x_1) + x_2 \tag{13}$$

where r is a random number between 0 and 1. It also assumes that the parent x_2 is not worse than x_1; that is, $f(x_2) \leq f(x_1)$.

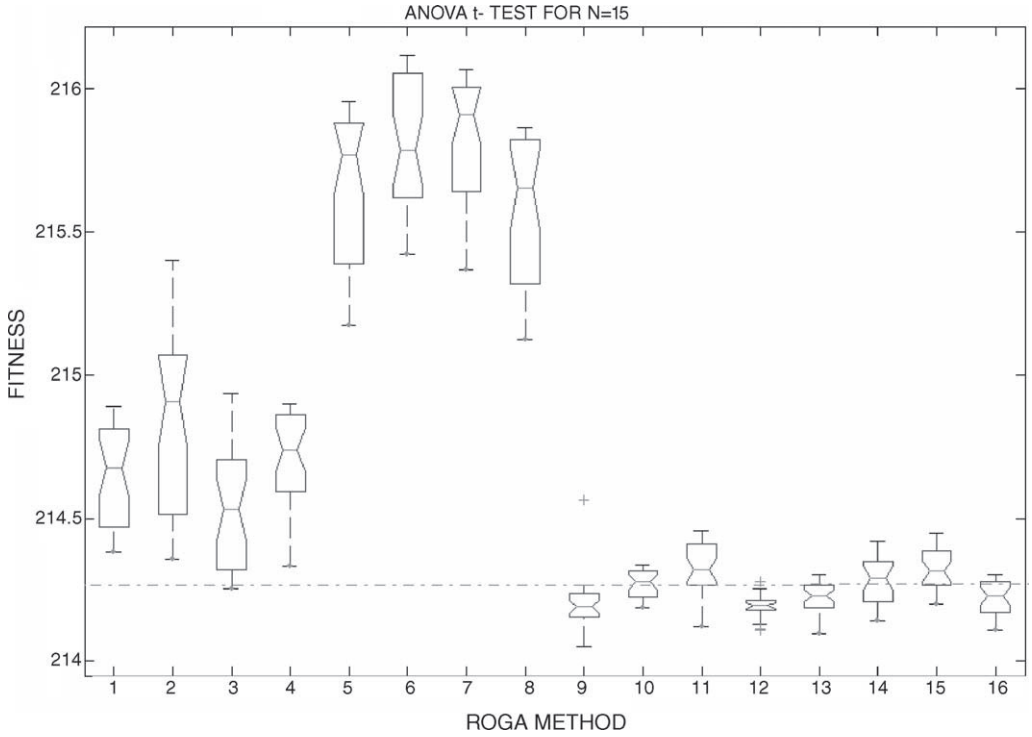

Fig. 5. ANOVA *t*-test for $N = 15$.

4.3.4. Hybrid cross-over methods (h-XO)

In this paper we design four hybrid cross-over methods from the three above said cross-over operations. Among the four hybrid methods, first three methods consist any of two methods with 50% of population size each where as the fourth hybrid method involves all the three cross-over methods.

Hybrid cross-over 1:	AMXO (50%) + ACXO (50%)
Hybrid cross-over 2:	AMXO (50%) + DBXO (50%)
Hybrid cross-over 3:	ACXO (50%) + DBXO (50%)
Hybrid cross-over 4:	AMXO (40%) + ACXO (30%) + DBXO (30%)

4.4. Mutation

The final genetic operator is mutation. It can create a new genetic material in the population to maintain the population's diversity. It is nothing but changing a random part of string representing the individual. In this paper, dynamic mutation is applied.

4.4.1. Dynamic mutation

Michalewicz [8] proposed this mutation operator also called non-uniform mutation. It is designed for fine-tuning capabilities aimed at achieving high precision. For a given parent x, if the element x_k is randomly selected from the following two possible choices:

$$x'_k = x_k + \Delta(t, x_k^{\mathrm{U}} - x_k) \quad \text{or}$$
$$x'_k = x_k + \Delta(t, x_k - x_k^{\mathrm{L}}) \tag{14}$$

The function $\Delta(t, \mathrm{d}x)$ returns a value in the range [0, $\mathrm{d}x$] such that the value of $\Delta(t, \mathrm{d}x)$ approaches 0 as t increases. This property causes the operator to search the space uniformly initially (when t is small) and very locally at later stages. The function $\Delta(t, \mathrm{d}x)$ is given as follows:

$$\Delta(t, \mathrm{d}x) = \mathrm{d}x\, r\left(1 - \frac{t}{T}\right)^d \tag{15}$$

where r is a random number from (0, 1), T the maximal generation number and d is a parameter determining the degree of non-uniformity (usually assumed as 2 or 3).

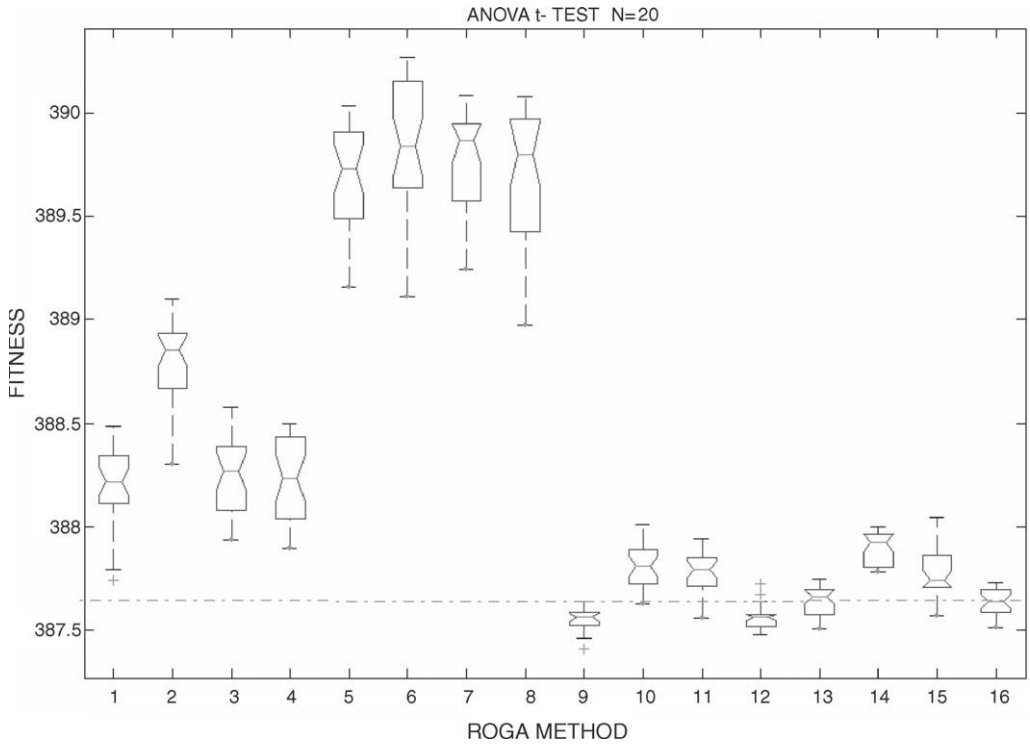

Fig. 6. ANOVA t-test for $N = 20$.

4.5. Elitism

In order to enrich the future generations with specific genetic information of the parent with best fitness from the current generations, that particular parent with the best fitness is preserved in the next generations. This method of preserving the elite parent is called *elitism*. This property is incorporated in our proposed algorithms.

5. Numeric example, simulation results and statistical analyses

To test the efficacy of our proposed algorithm, we consider the optimal control problem from Eqs. (5) and (6) with the following functions:

$$\theta_i(u_i) = \frac{1}{u_i} \quad \text{and} \quad \phi(x_i) = x_i^2 \tag{16}$$

Now Eq. (5) becomes,

$$\min_{u_1,\ldots,u_N} \left\{ J = \sum_{i=1}^{N} \left(\frac{1}{u_i} + x_i^2 \right) \right\} \tag{17}$$

subject to

$$x_i = \max(x_{(i-1)}, a_i) + u_i \tag{18}$$

The optimal controls (u_i) and cost or fitness (J) for the objective function given in Eq. (10) are computed with the following parameter settings.

The dimension or the number of jobs involved in the objective function $N = (5, 10, 15, 20, 25)$, cross-over probability $P_c = 0.8$ and probability of mutation $P_m = 0.1$. The maximum number of generations is set as 1000 with the population size of 50.

The arrival sequence (a_i for $i = 1$ to N) is {1, 1.2, 1.5, 1.8, 2, 2.2, 2.5, 2.8, 3, 3.2, 3.5, 3.8, 4, 4.2, 4.5, 4.8, 5, 5.2, 5.5, 5.8, 6, 6.2, 6.5, 6.8, 7}. The number of arrival times is taken according to the dimension of the problem, i.e., the number of jobs considered for processing.

With four different selections and four different cross-overs we consider 16 methods of the proposed

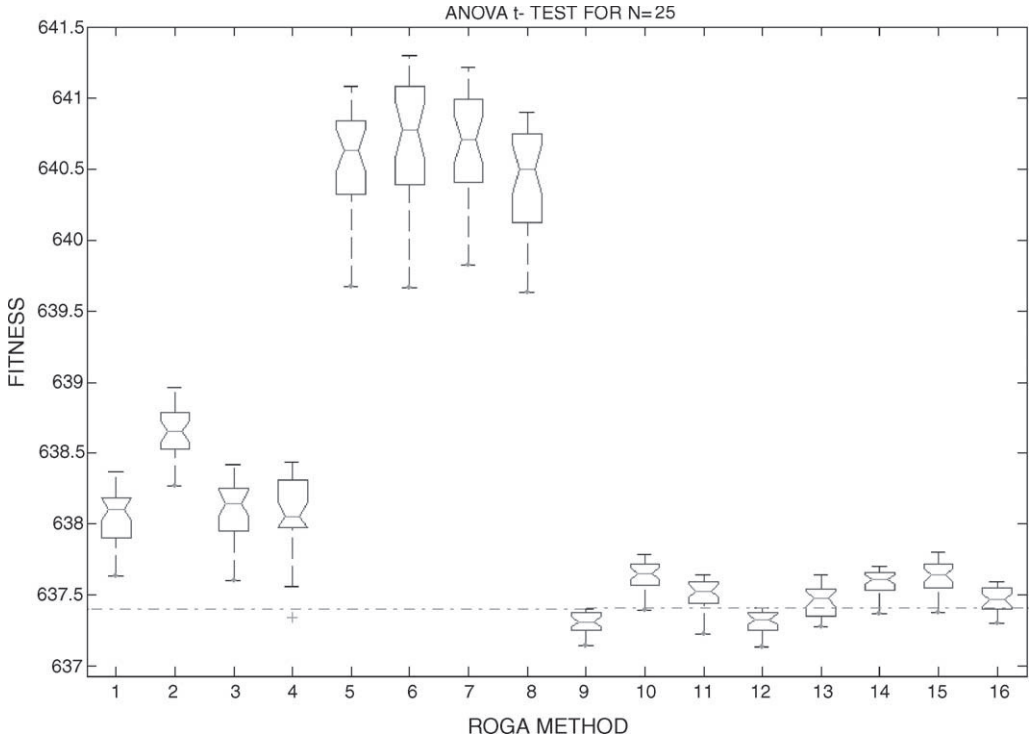

Fig. 7. ANOVA t-test for $N = 25$.

algorithm, each differs from the other in the selection method and the cross-over method adopted. Each of the four selection methods are combined with four cross-overs and a common dynamic mutation. Since mutation is carried out only for 10–15% of the total population, we consider non-uniform mutation or dynamic mutation (DM) only in our proposed algorithm. All the 16 different methods are given in Table 1.

Among the 16 methods, first eight methods are designed with either RWS or TS method and hybrid cross-over techniques. The remaining eight methods involve hybrid selection as well as hybrid crosover techniques.

All the 16 methods are executed by varying the number of jobs considered as 5, 10, 15, 20, 25. The fitness values for all the proposed methods are taken by running the simulation 500 times at different times. The *average values* (mean) and the *standard deviations* (S.D.) of the fitness values for each method are calculated. In order to strengthen the comparison, few more statistics tests are conducted, the *co-efficient variance*, which is calculated from the ratio of standard deviation to the mean and *the average deviation*, which gives the average of the absolute deviation of the fitness values from their mean, which are taken in 500 simulation runs. Added to these analyses, *hypothesis t-test and analysis of variance* (ANOVA) test also carried out to validate the efficacy among the proposed algorithms. These statistics analysis are presented in Tables 2–6. The graphical analyses are done through Box plot, which are shown in Figs. 3–7.

Table 7
Forward algorithm

Step 1: (initialization) $k = 1$, $n = 1$, $a_{N+1} = \propto$;
  while $n \leq N$ do

    Step 2: solve sub-optimal problem $Q(k, n)$;
    Step 3: (identify single busy periods)
    If $x_n^*(k, n) < a_{N+1}$ then
    $u_j^* \leftarrow u_j^*(k, n)$ for $j = k,\ldots, n$
    $k \leftarrow n + 1$;
    endif

    Step 4: (increment index n)
    $n \leftarrow n + 1$;

  end while

Table 8
Fitness values through forward algorithm

$N = 5$	34.8251
$N = 10$	102.4837
$N = 15$	226.2718
$N = 20$	407.2486
$N = 25$	661.8602

The fitness values of the same objective function, given in Eqs. (17) and (18), are calculated for $N = 5$, 10, 15, 20, 25 with the same arrival timings using the existing optimization algorithm, forward algorithm, which is given in Table 7. The results obtained through FA are presented in Table 8.

The percentage of deviation of the average fitness value of 16 proposed methods from the existing best optimal control method, forward algorithm for number of jobs varying from 5 to 25 is calculated using Eq. (19), for analysing the better performance of the proposed algorithm. They are presented in Table 9.

$$\%\ \text{of deviation of the fitness value} = \left(\frac{\text{fitness of FA} - \text{fitness of RGA}}{\text{fitness of RGA}}\right) \times 100 \qquad (19)$$

Table 9
The % of deviation of the Fitness value of RCGAs from FA

Method/Jobs	The % of deviation of the Fitness value from FA				
	$N = 5$	$N = 10$	$N = 15$	$N = 20$	$N = 25$
RCGA_a1	−0.00416	−0.22463	5.46652	4.91024	3.73031
RCGA_a2	−0.02419	−0.38572	5.31064	4.74630	3.63346
RCGA_a3	−0.02353	−0.24126	5.39364	4.88844	3.72322
RCGA_a4	−0.01082	−0.23398	5.46377	4.89927	3.72702
RCGA_b1	−0.13258	−0.30831	4.97639	4.51061	3.34717
RCGA_b2	−0.20750	−0.40724	4.85157	4.46582	3.32322
RCGA_b3	−0.19746	−0.31399	4.83182	4.48343	3.30837
RCGA_b4	−0.15780	−0.30938	4.95319	4.49629	3.31217
RCGA_c1	0.02271	0.26970	5.64060	5.08153	3.85323
RCGA_c2	0.01981	0.20645	5.57844	5.01134	3.79956
RCGA_c3	0.02023	0.22437	5.60178	5.02052	3.82021
RCGA_c4	0.02179	0.25557	5.63906	5.08088	3.85198
RCGA_d1	0.02156	0.25484	5.62418	5.05989	3.82843
RCGA_d2	0.01120	0.14826	5.57193	4.98810	3.79533
RCGA_d3	0.01679	0.20326	5.59490	5.02522	3.79952
RCGA_d4	0.02146	0.22781	5.62324	5.05900	3.82750

6. Discussion and conclusions

In order to compare the validity and usefulness of the proposed real coded GA method, 500 simulated results for each method are taken at different timings. In order to hasten the convergence in real coded GA's solution, the larger number of population (*Pop_Size*), generation (*max_gen*), are set to 50 and 1000, respectively. The performance of different algorithms is compared with respect to the solution accuracy in the fitness, the standard deviations, co-efficient variance, average deviation, ANOVA *t*-test, and the percentage of deviation in the fitness from the existing best method, forward algorithm. For the objective function of the hybrid model of a single stage manufacturing system, real coded GAs were applied and the mean best results obtained in 500 independent runs after 1000 generations are used for comparison.

From the results stated in Tables 2–6, it is obvious that the method *RCGA_c1* is the best (TS + RWS – with AMXO + ACXO and DM) followed by method *RCGA_c4* (TS + RWS – with AMXO + ACXO + DBXO and DM) and method *RCGA_d1* (TS + RWS + TS + RWS – with AMXO + ACXO and DM). This clearly establishes the fact that hybrid selections together with hybrid cross-over methods yield better solutions. This is the most significant outcome of the experiments performed. These combinations have been shown to work efficiently with regard to an optimal control problem here but it is believed that there might be equally efficient with regards to all other problems where real coded GA can be used.

The fitness regulation between the existing FA and the 16 proposed RCGA methods is calculated using Eq. (19) and presented in Table 9. Out of the 16 proposed methods, the first eight methods (methods 1–8) do not adopt the hybrid selection procedure but only the hybrid cross-over techniques where as the next eight methods (methods 9–16) adopt both the hybrid selection and the hybrid cross-over techniques. For a method to be effective this percentage of deviation, which is also known as fitness regulation, should be positive and as large as is really possible. From Table 9 it is observed that the negative values of fitness regulation indicate that the methods 1–8 are clearly inferior to FA only with regard to smaller number of jobs. But when the number of jobs considered is increased, these methods (methods 1–8) produce better solution than FA, which is clearly seen from the positive fitness regulation values. The remaining methods (methods 9–16) produce better solution for small number of jobs as well as higher number of jobs. Based on this, it can be concluded that the hybrid real coded GA with the hybrid genetic operators gives the best optimal value in comparison with the other methods, including the proposed methods 1–8 and also the existing forward algorithm method.

The hypothesis *t*-test and ANOVA test are also carried out to prove the best method between RCGA and their *P*-values are also recorded. The ANOVA test results are drawn using Box plots. From the box plot figures, which are given in Figs. 3–7, it is obvious that, method RCGA_c1 gives the best results among all other RCGA methods. To ease graphical viewing, a dot-dash line is depicted with reference to method RCGA_c1, and it is observed that all other methods exceed that reference line. This clearly indicates that the hybrid selection with the hybrid combination of AMXO and ACXO gives the best results.

In other words, the superior performance of the hybrid combinations of the genetic operators for real coded genetic algorithms has been clearly established in this paper.

References

[1] C.G. Cassandras, D.L. Pepyne, Y. Wardi, Optimal control of a class of hybrid systems, IEEE Trans. Automat. Cont. 46 (2001) 398–415.
[2] P. Zhang, C. Cassandras, An improved forward algorithm for optimal control of a class of hybrid systems, IEEE Trans. Automat. Cont. 47 (2002) 1735–1739.
[3] D.L. Pepyne, C.G. Cassandras, Modeling, analysis, and optimal control of a class of hybrid systems, Discrete Event Dynamic Syst.: Theory Appl. 8 (1998) 175–201.
[4] Y. Wardi, C.G. Cassandras, D.L. Pepyne, A backward algorithm for computing optimal controls for a single stage hybrid manufacturing systems, Int. J. Prod. Res. 39 (2) (2002) 369–394.
[5] Y.C. Cho, C.G. Cassandras, D.L. Pepyne, Forward decomposition algorithms for optimal control of a class of hybrid systems, Int. J. Robust Nonlinear Control 11 (5) (2001) 497–513.
[6] Y.C. Cho, C.G. Cassandras, Optimal control of steel annealing processes as hybrid systems, in: Proceedings of the 39th IEEE Conf. on Decision and Control, 2000.
[7] D. Goldberg, Genetic algorithms in search, in: Optimisation and Machine Learning, Addison Wesley, Massachusetts, USA, 1989.

[8] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, 1994.
[9] S. Baskar, P. Subbaraj, M.V.C. Rao, Performance of hybrid real coded genetic algorithms, Int. J. Comput. Eng. Sci. 2 (4.) (2001).
[10] L. Davis, A Hand Book of Genetic Algorithm, New York, 1990.
[11] H. Schwefel, Numerical Optimization of Computer Models, John Wiley & Sons, Chichester, 1981.
[12] M.S. Arumugam, M.V.C. Rao, Novel hybrid approaches for real coded genetic algorithm to compute the optimal control of a single stage hybrid manufacturing systems, Int. J. Comput. Intell., in press.