Jurnal Teknik Industri, Vol. 14, No. 1, Juni 2012, 1-12
ISSN 1411-2485 print / ISSN 2087-7439 online

A New Evolutionary Algorithm Based on Bacterial Evolution and Its Application for Scheduling a Flexible Manufacturing System

Chandramouli Anandaraman[1]*, Arun Vikram Madurai Sankar[1], Ramaraj Natarajan[1]

Abstract: A new evolutionary computation algorithm, Superbug algorithm, which simulates evolution of bacteria in a culture, is proposed. The algorithm is developed for solving large scale optimization problems such as scheduling, transportation and assignment problems. In this work, the algorithm optimizes machine schedules in a Flexible Manufacturing System (FMS) by minimizing makespan. The FMS comprises of four machines and two identical Automated Guided Vehicles (AGVs). AGVs are used for carrying jobs between the Load/Unload (L/U) station and the machines. Experimental results indicate the efficiency of the proposed algorithm in its optimization performance in scheduling is noticeably superior to other evolutionary algorithms when compared to the best results reported in the literature for FMS Scheduling.

Keywords: Superbug algorithm, FMS, scheduling, makespan, bacterial evolution.

Introduction

Over the years, researchers suggest that nature is a great source for inspiration to both develop intelligent systems as well as provide solutions to complex problems. Evolutionary pressure forces living organisms to develop great expertise in fighting for food, territories and mates. Many of these skills and techniques have been imitated to develop optimization algorithms, and the evolution over a number of generations enhances the performance of an algorithm. In this work, a new nature-inspired algorithm, based on bacterial evolution, Superbug algorithm is proposed.

The algorithm is inspired by the behaviour of bacteria such as Staphylococcus aureus, Streptococcus, Enterococcus, Salmonella and E. coli. Several antibiotics were developed to cure infections by these microorganisms in the early 20th century. It was discovered that many of these bacteria had developed resistance to the antibiotics over time. The increasing levels of antibiotic resistance and the emergence of epidemic strains of bacterial pathogens over the last decade (Enright *et al.* [7] and Livermore [9]) highlight the adaptability of bacteria and the remarkable speed of bacterial evolution. In the face of constant environmental challenges, the ability of bacteria to generate genetic variation is crucial for their survival. Bacterial genomes tend to evolve through several routes: Mutation to existing genes, DNA loss or rearrangement or horizontal transfer of genes from one bacterium to another (Ziebuhr *et al.* [17]).

[1] Department of Production Engineering, National Institute of Technology, Tiruchirappalli 620 015, India.
Email: acmouli89@gmail.com

*Corresponding author

Evolutionary computation has been applied to the scheduling of multiple CPU cores on a parallel computer, and has recreated known scheduling algorithms (Jaros *et al.* [8]). Of recent interest are bio-inspired approaches to manufacturing system design, including biomimetics, where computational systems mimic behaviours found in natural organisms. Examples of biomimetic approaches include mimicking the social behaviour of insect colonies (Truszkowski *et al.* [15]), flocking (Spector *et al.* [12] and Anthony [2]), and using the concept of chemotaxis to facilitate robust network routing. Two bio-inspired studies include the recent works (Babaoglu *et al.* [3, 4] and Patel *et al.* [10]) inspired by the synchronization of firefly flashes. Swarm intelligence algorithms such as Bee algorithm were applied to schedule jobs for a machine (Pham *et al.* [11]). Chong *et al.* [6] utilized an efficient neighbourhood structure to search for feasible solutions and iteratively improve on prior solutions.

Bilge and Ulusoy [5] presented a model for simultaneous scheduling of machines and material handling system in an FMS. This problem was approached using Genetic algorithms (GA) (Ulusoy *et al.* [16]). A new hybrid genetic algorithm composed of GA and heuristic for the simultaneous scheduling problem for minimization of makespan was presented (Abdelmaguid *et al.* [1]). A number of evolutionary techniques have been applied for scheduling different elements in an FMS, making it a convenient problem for testing new solution methodologies.

The optimization of FMS schedules using non-traditional techniques was performed (Sreedhar Kumar *et al.* [13]) The problem of simultaneous scheduling of machines and two identical automated guided vehicles (AGVs) in an FMS using Sheep

Flock Heredity algorithm was addressed (Subbaiah *et al.* [14]). In the current work, a new evolutionary algorithm based on bacterial evolution namely, Superbug algorithm is proposed in order to solve optimization problems. The algorithm is designed to enhance the diversity of the search domain by combining effective local and global search procedures. This helps in obtaining high quality solutions compared to other existing optimization algorithms found in the literature.

Methods

Superbug Algorithm

The primary mode through which a bacterium develops antibiotic resistance is through mutation. After mutation, a different gene sequence is generated, which attempts to survive the chemical action of the antibiotic. While this is similar to the evolution of other organisms, another technique used by bacteria is the Lateral gene transfer. In this, genes are transferred from one bacterium to another by cell to cell contact. This helps more bacteria acquire a drug resistant gene and enhance its chances of survival. The ability of the transformed bacterium is further improved by single point mutation, i.e. transferring a vulnerable portion of a gene to another location in the same gene sequence.

The resulting bacteria have better resistance to the antibiotic. They reproduce and multiply their numbers. After going through these stages of evolution several times, a bacterium would have accumulated several antibiotic-resistant genes. Such a bacterium is called a superbug. This evolution process has the capability to generate a large number of bacteria with high antibiotic resistance compared to the present GA procedures.

Stages in the Evolution of a Bacterium

Genetic Mutation

When the bacterium is exposed to the antibiotic, its normal metabolic processes are suppressed. This forces it to try to survive by mutating itself. Mutation rearranges part of the gene sequence so that it becomes immune to the antibiotic. The probability of a bacterium undergoing mutation depends on the levels of exposure to antibiotic and the extent of suppression of its metabolic activities. The mutation takes place in two levels, inverse mutation and pairwise interchange mutation (Subbaiah *et al.* [14]).

Lateral Gene Transfer

After undergoing mutation, a population of bacteria with varying levels of fitness (in terms of resistance to the antibiotic) emerges. A bacterium with higher fitness tries to increase its fitness by engaging contact with another bacterium having a lower fitness value. A random gene is transferred from the bacterium having lower fitness to the bacterium with higher fitness. This helps in transferring some of the drug resistant strains from one bacterium to another, thereby improving the fitness level considerably.

Single Point Mutation

This technique is employed by the bacterium in the final stage of its evolution. A single gene is transferred to a random location in the bacterium itself, i.e. mutation takes place in a single point. This is called single point mutation or point mutation.

Reproduction

The set of bacteria which have acquired drug resistance tend to survive and reproduce. The next generation of bacteria, if exposed to the antibiotic, will employ the same techniques to further improve their resistance. After several generations, a set of bacteria with resistance to several drugs is developed, which is called a superbug.

In the current work, the algorithm is tested on a FMS scheduling problem. The objective is to minimise the makespan which is taken as the fitness function.

FMS Description

An FMS is considered in which there are four machines having Computer Numerical Control machines (CNCs), each with an independent and self-sufficient tool magazine, one Automatic Tool Changer (ATC) and one Automatic Pallet Changer (APC) and with multiple Automated Guided Vehicles (AGVs) as material handling devices. The problem of simultaneous scheduling of machines and AGVs is addressed. We have considered 4 different layouts and 10 job sets consisting of 1-8 different job sets and operations on machines to be performed.

The objective is to minimize makespan. An iterative procedure is developed in which a new machine schedule is generated using the superbug algorithm after each iteration. Figure 1 illustrates the different phases applied to every generated sequence.

Assumptions

The types and number of machines are known, there is sufficient input/output buffer space for each machine's machine loading allocation of tools to machine assignment of operation to machine are

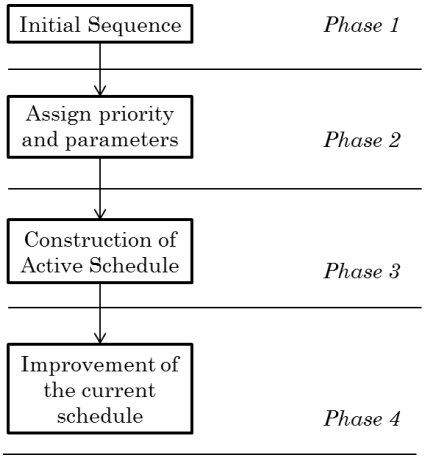

Figure 1. Phases of the given approach

made pallet and other necessary equipment are allocated. AGVs are used for material transfer between machines and also between L/U station and machines. The speed of AGV (40 m/min), the distance between the two machines and the distance between loading/ unloading machines are known.

Input Data

The input data that is, travelling time matrix and the layouts taken from Subbaiah *et al.* [14] are shown in Tables 1-4 and job sets for the problem taken from Bilge and Ulusoy [5] are shown in Table 5. The L/U station serves as a distribution centre for parts not yet processed and as a collection centre for parts finished. All vehicles start from the L/U station initially.

Scheduling of FMS

Machines are scheduled based on the operation sequence derived by the algorithm. Initially one of the AGVs carries jobs from the L/U station to the respective workstations where the first operations are scheduled. For subsequent operations, whichever AGV is available reaches the machine where the previous operation has been completed, picks up the job and carries it to the machine scheduled for the next operation. The flowchart given in Figure 2 shows the scheduling methodology adopted for this problem. The equations used for computing the job completion time and makespan are given in Eqs. (1) - (3).

$$O_{ij} = T_{ij} + P_{ij} \quad (1)$$

Job completion time: $C_i = \sum_{j=1}^{n} O_{ij}$ (2)

Makespan: $\max(C_1, C_2, C_3, \dots)$ (3)

Table 1. Travel time matrix for Layout 1

		Destination				
		L/U	M1	M2	M3	M4
Source	L/U	0	6	8	10	12
	M1	12	0	6	8	10
	M2	10	6	0	6	8
	M3	8	8	6	0	6
	M4	6	10	8	6	0

Table 2. Travel time matrix for Layout 2

		Destination				
		L/U	M1	M2	M3	M4
Source	L/U	0	4	6	8	6
	M1	6	0	2	4	2
	M2	8	12	0	2	4
	M3	6	10	12	0	2
	M4	4	8	10	12	0

Table 3. Travel time matrix for Layout 3

		Destination				
		L/U	M1	M2	M3	M4
Source	L/U	0	2	4	10	12
	M1	12	0	2	8	10
	M2	10	12	0	6	8
	M3	4	6	8	0	2
	M4	2	4	6	12	0

Table 4. Travel time matrix for Layout 4

		Destination				
		L/U	M1	M2	M3	M4
Source	L/U	0	4	8	10	14
	M1	18	0	4	6	10
	M2	20	14	0	8	6
	M3	12	8	6	0	6
	M4	14	14	12	6	0

where

i : job number

j : operation number

n : number of job

O_{ij} : time needed for j-th operation of i-th job

T_{ij} : total traveling time for i-th job before j-th operation

P_{ij} : total processing time for i-th job before j-th operation

Table 5. Job set data

Job Set 1						Job Set 6				
Job 1	M1(8)	M2(16)	M4(12)			Job 1	M1(9)	M2(11)	M4(7)	
Job 2	M1(20)	M3(10)	M2(18)			Job 2	M1(19)	M2(20)	M4(13)	
Job 3	M3(12)	M4(8)	M1(5)			Job 3	M2(14)	M3(20)	M4(9)	
Job 4	M4(14)	M2(18)				Job 4	M2(14)	M3(20)	M4(9)	
Job 5	M3(10)	M1(15)				Job 5	M1(11)	M3(16)	M4(8)	
Job Set 2						Job Set 7				
Job 1	M1(10)	M4(18)				Job 1	M1(6)	M4(6)		
Job 2	M2(10)	M4(18)				Job 2	M2(11)	M4(9)		
Job 3	M1(10)	M3(20)				Job 3	M2(9)	M4(7)		
Job 4	M2(10)	M3(15)	M4(12)			Job 4	M3(16)	M4(7)		
Job 5	M1(10)	M2(15)	M4(12)			Job 5	M1(9)	M3(18)		
Job 6	M1(10)	M2(15)	M3(12)			Job 6	M2(13)	M3(19)	M4(6)	
						Job 7	M1(10)	M2(9)	M3(13)	
						Job 8	M1(11)	M2(9)	M4(8)	
Job Set 3						Job Set 8				
Job 1	M1(16)	M3(15)				Job 1	M2(12)	M3(21)	M4(11)	
Job 2	M2(18)	M4(15)				Job 2	M2(12)	M3(21)	M4(11)	
Job 3	M1(20)	M2(10)				Job 3	M2(12)	M3(21)	M4(11)	
Job 4	M3(15)	M4(10)				Job 4	M2(12)	M3(21)	M4(11)	
Job 5	M1(8)	M2(10)	M3(15)	M4(17)		Job 5	M1(10)	M2(14)	M3(18)	M4(9)
Job 6	M2(10)	M3(15)	M4(8)	M1(15)		Job 6	M1(10)	M2(14)	M3(18)	M4(9)
Job Set 4						Job Set 9				
Job 1	M4(11)	M1(10)	M2(7)			Job 1	M3(9)	M1(12)	M2(9)	M4(6)
Job 2	M3(12)	M2(10)	M4(8)			Job 2	M3(16)	M2(11)	M4(9)	
Job 3	M2(7)	M3(10)	M1(9)	M3(8)		Job 3	M1(21)	M2(18)	M4(7)	
Job 4	M2(7)	M4(8)	M1(12)	M2(6)		Job 4	M2(20)	M3(22)	M4(11)	
Job 5	M1(9)	M2(7)	M4(8)	M2(10)	M3(8)	Job 5	M3(14)	M1(16)	M2(13)	M4(9)
Job Set 5						Job Set 10				
Job 1	M1(6)	M2(12)	M4(9)			Job 1	M1(11)	M3(19)	M2(16)	M4(13)
Job 2	M1(18)	M3(6)	M2(15)			Job 2	M2(21)	M3(16)	M4(14)	
Job 3	M3(9)	M4(3)	M1(12)			Job 3	M3(8)	M2(10)	M1(14)	M4(9)
Job 4	M4(6)	M2(15)				Job 4	M2(13)	M3(20)	M4(10)	
Job 5	M3(3)	M1(9)				Job 5	M1(9)	M3(16)	M4(18)	
						Job 6	M2(19)	M1(21)	M3(11)	M4(15)

Table 6. Numbering of operations in Job Set 1

Job No.	Job 1			Job 2			Job 3			Job 4			Job 5
Machine	M1	M2	M4	M1	M3	M2	M3	M4	M1	M4	M2	M3	M1
Operation No.	1	2	3	4	5	6	7	8	9	10	11	12	13

Table 7. Calculation of makespan

Job	Machine	AGV	Travel time	Job reach	Job ready	Job completion
3,1	3	1	10	10	10	22
2,1	1	2	6	6	6	26
1,1	1	2	18	18	26	34
5,1	3	1	18	18	22	32
3,2	4	1	6	28	28	36
3,3	1	1	10	46	46	61
2,2	3	2	8	34	34	44
1,2	2	2	14	48	48	64
5,2	1	1	16	62	62	77
4,1	4	2	22	70	70	84
1,3	4	1	14	76	84	96
4,2	2	2	8	92	92	110
2,3	2	1	12	96	96	114

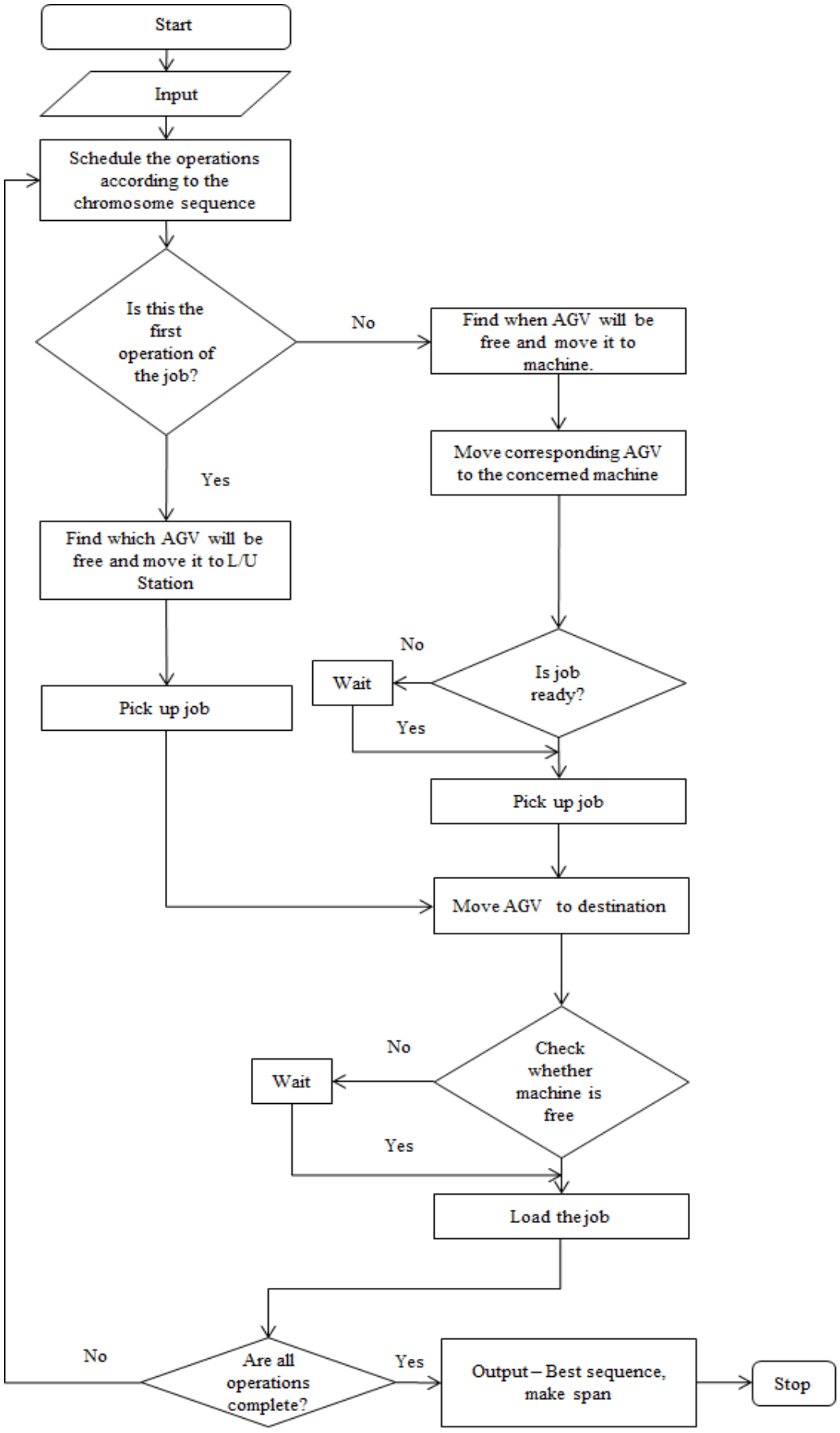

Figure 2. Scheduling flowchart

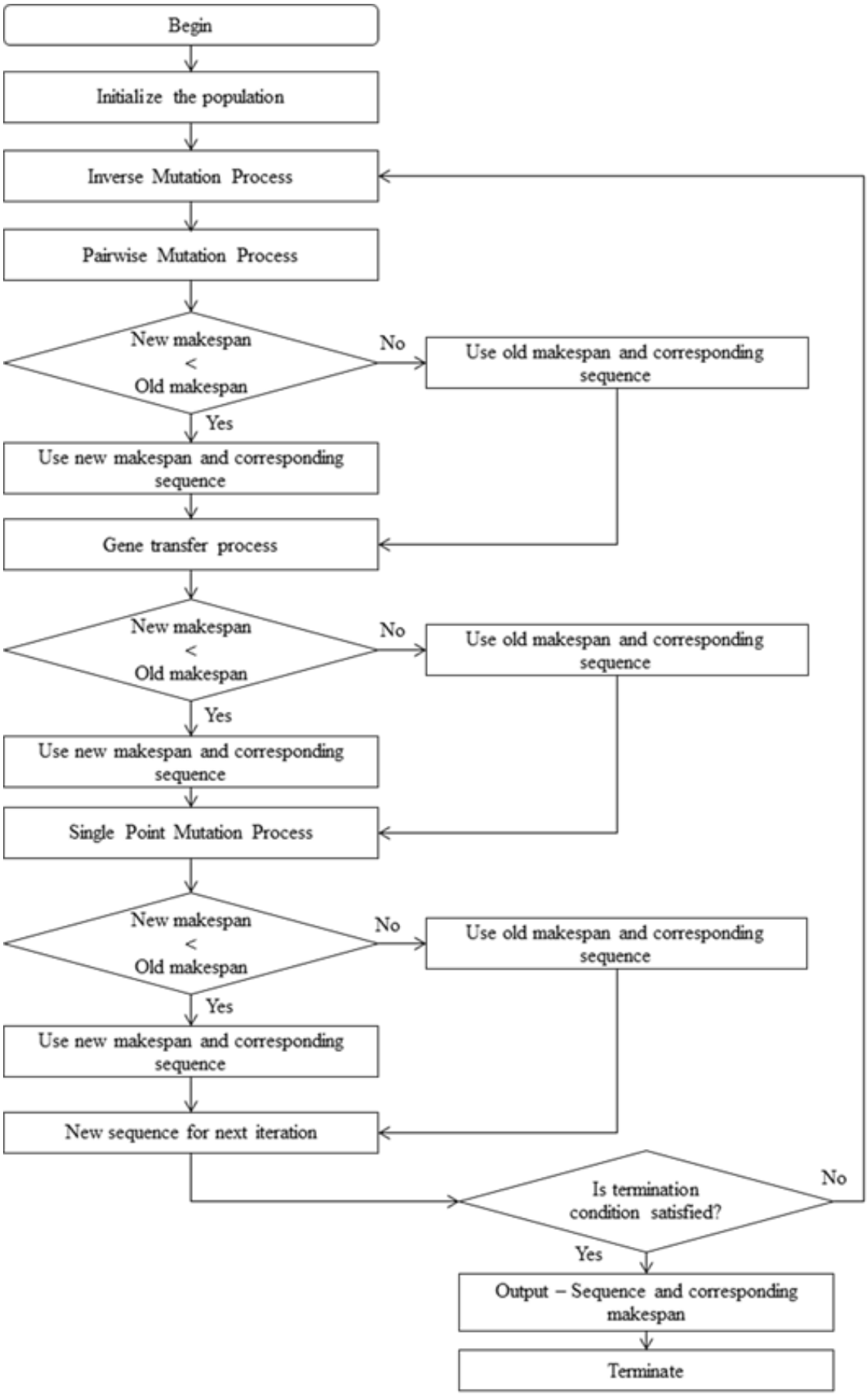

Figure 3. Flowchart for Superbug algorithm

Table 8. Initial sequences

Sequence													Makespan
1	2	7	4	12	3	8	5	10	6	11	9	13	125
7	1	10	12	8	4	2	13	3	11	5	9	6	114
7	1	10	2	11	12	4	8	3	13	5	9	6	122
4	5	10	11	12	1	2	7	8	9	13	6	3	134
7	4	1	12	8	9	2	10	5	13	6	11	3	118
4	7	8	5	10	6	11	1	2	9	12	3	13	139
10	7	12	1	2	4	13	8	5	3	11	9	6	110
4	5	7	6	8	1	10	2	9	3	11	12	13	153
7	1	4	12	2	8	10	9	5	13	6	11	3	102
4	5	1	6	12	13	7	10	11	2	3	8	9	157

For example, Job set1 and Layout 1 are considered for scheduling. For scheduling the FMS and calculating the makespan, initially continuous numbers are given for each of the operations as shown in Table 6. These numbers are used to generate 10 initial random sequences, while obeying the precedence relation, i.e., the operations of a particular job must be in increasing order.

Calculation of Makespan for a Particular Operation Sequence

The following initial sequence is generated randomly, using the numbers assigned to the operations in Table 6. The processing time and travelling time matrix shown in Table 1-5.

7 4 1 12 8 9 5 2 13 10 3 11 6

The makespan of the job set when scheduled according to the above sequence is calculated as given in Table 7. The first column denotes the Job number and the operation for that job needed to find the machine and processing time for that operation. The sequence of these operations is determined by the string of numbers generated by the algorithm. The second column indicates the machine used for this Job and operation, as given in Table 5. The third column specifies the AGV assigned to transport this job. This is determined by computing the time each AGV will take to reach the job and deliver it to its destination, based on its previous assignment and the travel times between machines (Tables 1, 2, 3 & 4). The AGV which could deliver the job at the earliest, based on this calculation, is assigned. The column 'Travel time' denotes the time taken by the AGV to deliver the job while the column 'Job reach' specifies the time the job reaches its intended machine. 'Job ready' shows the time the job is taken up for processing (same as 'Job reach' if the machine is free, else some delay occurs). 'Job completion' is the time the job will get completed and be ready for next operation. This value is arrived at by adding the processing time (Table 5) to the 'Job ready' time.

Implementation of Superbug Algorithm for Scheduling an FMS

Flowchart of Superbug Algorithm

Figure 3 shows the flow chart of the proposed Superbug algorithm. The first step is initializing a population composed of randomly generated individuals covering a wide range of solution spaceand measuring the fitness of the individuals. The subsequent stages include the mutation and the gene transfer where unfit individuals get replaced by fit ones. These operations are repeated for several generations until the termination criteria are met.

Steps in Superbug Algorithm

The algorithm consists of the following steps:
1. Generation of initial population
2. Mutation of the bacteria (inverse and pairwise interchange mutation)
3. Gene transfer between bacteria to enhance fitness
4. Single point mutation of the modified bacteria

Generation of Initial Population

A set of ten initial sequences are randomly generated, as given in Table 8.

Mutation

The mutation consists of two steps, inverse mutation and pairwise interchange mutation:

(a) Inverse mutation
In a sequence, two positions i and j are randomly selected. The portion of the sequence between these two positions is inverted to get a new mutated sequence. The new sequence represents the sequence of operations after mutation. If the makespan of the mutated sequence is less than the makespan of the original sequence, the old sequence is replaced by the new sequence.

Original sequence:
4 **5** 7 8 9 12 10 1 11 13 2 **6** 3

Mutated sequence:
4 **6 9 8 7 13 11 2 10 12 1 5** 3

(b) Pairwise interchange mutation
Two positions are i and j randomly selected in the sequence. The operations in these positions are interchanged to obtain the mutated sequence. The makespan of the new sequence is compared with the makespan of the parent sequence. The sequence

having the lower makespan value is stored and used for next stage operation.

Original sequence

4 **5** 7 8 9 12 10 1 **11** 13 2 6 3

Mutated sequence

4 **11** 7 8 9 12 10 1 **5** 13 2 6 3

Mutation at positions 2 and 9

Gene Transfer

Two sequences are needed for gene transfer. The sequence having the best makespan (lowest) in the given population is chosen as donor. The sequence with the second best makespan is the receptor. The selection is done this way to maximize the chances of the modified sequence giving an improved makespan. A random set of operations is identified in the donor sequence. The order of operations in the identified set is used to replace the same operations in the other sequence. The resulting sequence replaces the original sequence if the makespan of the resulting sequence is less than that of the previous one.

Donor sequence

1 2 7 **4** 12 3 8 5 10 6 **11** 9 13

Receptor sequence:

4 5 10 **11** 12 1 2 7 8 9 13 6 3

Gene 4 8 11 is transferred to the receptor sequence and replaces the gene 4 11 8

Sequences after transfer:

Donor sequence

1 2 7 **4** 12 3 8 5 10 6 **11** 9 13

Receptor sequence:

4 5 10 8 12 1 2 7 **11** 9 13 6 3

Single Point Mutation

A random operation is selected in the sequence and moved to another random position in the sequence. If the makespan of the resulting sequence is less than that of the previous one, it replaces the previous sequence.

Original sequence

10 7 12 1 2 **4** 13 8 **5** 3 11 9 6

Mutation of operation at position 6 to position 10

Mutated sequence:

10 7 12 1 2 13 8 5 3 **4** 11 9 6

The set of sequences obtained from the above operations is sorted by their makespan values. The lowest makespan in the set is taken as the makespan of the generation. The sequences are again subjected to the operations till the convergence criterion is met.

Table 9. Parameter analysis

Parameter	Values taken for analysis	Best value	Corresponding best makespan
S	5, 10, 15, 20	5	91
N	500, 1000, 1500, 2000	1000	92
P_{ip}	0.05, 0.1, 0.15, 0.2	0.2	92
L_g	3, 4, 5, 6, 7	3	92
P_g	0.5, 0.6, 0.7, 0.8, 0.9	0.9	92
P_s	0.1, 0.2, 0.3, 0.4, 0.5	0.4	92

Selection of Parameters for Superbug Algorithm

The algorithm has the following parameters to be defined.

(1) Size of bacteria population (S)
(2) No. of generations for which the algorithm is applied (N)
(3) Probability of inverse mutation and pairwise interchange mutation (P_{ip})
(4) Length of gene transferred from one bacterium to another (L_g)
(5) Probability of gene transfer taking place (P_g)
(6) Probability of bacteria undergoing single point mutation (P_s)

An analysis is performed by assigning a set of values to each of the above parameters. The performance of the algorithm under each set of parameters is examined to arrive at an appropriate set of parameters that will provide optimal solutions to the problems considered.

The population size is important here as the bacteria will transfer genes to one another to improve their fitness. Keeping the population too low will give insufficient diversity of solutions, and a population too high will reduce the chances of constructive gene transfer. The number of generations is chosen in such a way that the results produced are reasonable. Probabilities of the inverse and pairwise mutations are kept low so that not too many unfit sequences are produced. Otherwise, due to the precedence restrictions in scheduling many of the mutated sequences will become unfit. For single point mutation, the probability is higher than inverse and pairwise mutation since the mutation is self-induced.

Table 9 shows the results of the analysis done on the parameters of the Superbug algorithm.

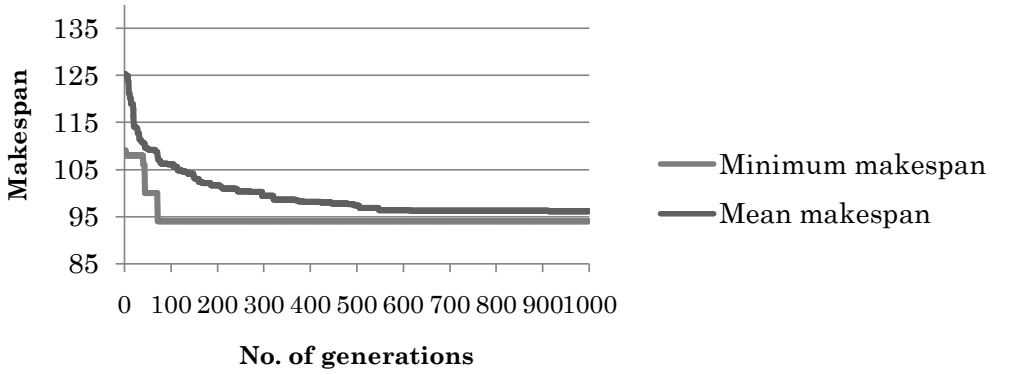

Figure 4. Convergence graph of makespan

Table 10. Results

Layout	Job Set	STW	UGA	AGA	PGA	SFHA	Superbug
1	1	96	96	96	96	90	92
	2	105	104	102	100	96	94
	3	105	105	99	99	105	96
	4	118	116	112	112	119	111
	5	89	87	87	87	87	86
	6	120	121	118	118	118	94
	7	119	118	115	111	128	113
	8	161	152	161	161	137	117
	9	120	117	118	116	111	105
	10	153	150	147	147	148	129
2	1	82	82	82	82	80	73
	2	80	76	76	76	76	66
	3	88	85	85	85	74	71
	4	93	88	88	67	96	88
	5	69	69	69	69	72	65
	6	100	98	98	98	86	79
	7	90	85	79	79	87	74
	8	151	142	151	151	128	97
	9	104	102	104	102	93	87
	10	139	137	136	135	130	102
3	1	84	84	84	84	80	78
	2	86	86	86	86	80	78
	3	86	86	86	86	79	76
	4	95	91	89	89	92	86
	5	76	75	74	74	73	70
	6	104	104	104	103	86	80
	7	91	88	86	83	94	83
	8	153	143	153	153	130	98
	9	110	105	106	105	94	93
	10	143	143	141	139	127	116
4	1	108	103	103	103	101	100
	2	116	113	108	108	113	104
	3	116	113	111	111	115	109
	4	126	126	126	126	130	125
	5	99	97	96	96	96	95
	6	120	123	120	120	125	103
	7	136	128	127	126	145	124
	8	163	163	163	163	146	136
	9	125	123	122	122	126	115
	10	171	164	159	158	173	145

Figure 4 shows the minimum makespan and mean makespan given by the algorithm when run with the above parameters.

Results and Discussions

The Superbug algorithm is used for generating optimal schedules for each problem by minimizing the makespan. The coding has been developed using MATLAB. A series of trial experiments are used to estimate the optimal parameters for the algorithm. Every instance of a problem is executed for 10 runs. Any alteration of the above parameters led to convergence at higher objective values than while using the optimal parameters. From the convergence graph for the algorithm (Figure 4) it is observed that, even as the minimum makespan is obtained in few generations, the mean makespan converges only after several hundred generations. This indicates the diversity of the solutions in the search space.

The results obtained from the algorithm for different problem sets are given in Table 10. The values of makespan obtained at convergence are compared with different algorithms. From Table 10, 36 out of 40 problems give better results using Superbug algorithm when compared with other standard algorithms such as Sliding Time Window (STW), Abdelmaguid Genetic Algorithm (AGA), Ulusoy Genetic Algorithm (UGA), Proposed Genetic Algorithm (PGA) by Subbaiah *et al.*[14] and Sheep Flock Heredity Algorithm (SFHA)

Conclusion

The proposed algorithm is found to be robust for the test problems and performs exceedingly well in majority of the problems considered. The algorithms present a good number of diversified solutions for the set of problems considered. The diversity in the set of solutions after every generation is preserved by a combination of local and global search procedure. The main contribution of this work is to prove the superiority of solutions for the given scheduling problem found by this algorithm as compared to other optimization algorithms.

The future research includes application of this algorithm for single or multiple objectives case considering different criteria like mean flow time, total tardiness, and maximum tardiness. The proposed algorithm can be applied to the scheduling problems in various manufacturing systems such as cellular manufacturing. Furthermore, it is possible that the proposed algorithm can be applied to optimize multi machine environment and the dynamic JSSP.

References

1. Abdelmaguid, T. F., Nassef, A. O., Kamal, B. A., and Hassan, M. F., A Hybrid GA/Heuristic Approaches to the Simultaneous Scheduling of Machines and Automated Guided Vehicles, *International Journal of Production Research*, 42, 2004, pp. 267-281.
2. Anthony, R. J., Emergence: A Paradigm for Robust and Scalable Distributed Applications, *Proceedings of the International Conference on Automatic Computing*, 2004, pp. 132-139.
3. Babaoglu, G., Canright, A., Deutsch, G. A. D., Caro, F., Ducatelle, L. M., Gambardella, N., Ganguly, M., Jelasity, R., Montemanni, A., and Urnes, T., Design Patterns from Biology for Distributed Computing, *ACM Transactions on Autonomous and Adaptive Systems*, 1(1), 2006, pp. 26–66.
4. Babaoglu, G., Binci, T., Jelasity, M., and Montresor, A., Firefly-Inspired Heartbeat Synchronization in Overlay Networks, *Proceedings of Self-Adaptive and Self-Organizing Systems* (SASO), 2007, pp. 77–86.
5. Bilge, U., and Ulusoy, G., A Time Window Approach to Simultaneous Scheduling of Machines and Material Handling System in an FMS, *Operations Research*, 43(6), 1995, pp. 1058-1070.
6. Chong, C. S., Malcolm Low, Y. H., Sivakumar, A. I., and Gay, K. L., Using a Bee Colony Algorithm for Neighborhood Search in Job Shop Scheduling Problems, In: *21st European Conference on Modeling and Simulation* (ECMS 2007).
7. Enright, M. C., Robinson, D. A., Randle, G., Feil, E. J., Grundmann, H., and Spratt, B. G., The Evolutionary History of Methicillin Resistant Staphylococcus Aureus (MRSA), *Proceedings of National Academy of Sciences, USA*, 2002, pp. 7687–7692.
8. Jaros, J., Ohlidal, M., and Dvorak, V., An Evolutionary Approach to Collective Communication Scheduling, In *Proceedings of the Conference on Genetic and Evolutionary Computation* (GECCO), 2007, pp. 2037–2044.
9. Livermore, D. M., Bacterial Resistance: Origins, Epidemiology, and Impact, *Clinical Infectious Diseases*, 36, 2003, pp. S11–S23
10. Patel, A., Degesys, J., and Nagpal, R., Desynchronization: The Theory of Self-Organizing Algorithms for Round Robin Scheduling, *Proceedings of Self-Adaptive and Self-Organizing Systems* (SASO), 2007, pp. 87–96.
11. Pham, D. T., Koc, E., Lee, J., and Phrueksanant, J., Using the Bees Algorithm to Schedule Jobs for a Machine, *Proceedings of Eighth Interna-*

tional Conference on Laser Metrology, CMM and Machine Tool Performance, 2007, pp. 430–439.

12. Spector, L., Klein, J., Perry, C., and Feinstein, M., Emergence of Collective Behavior in Evolving Populations of Flying Agents, *Genetic Programming and Evolvable Machines*, 6(1), 2005, pp. 111–125.
13. Sreedhar Kumar, A. V. S., Veeranna, V., Durga Prasad, B., and Dattatraya Sarma, B., Optimization of FMS Schedules Using Non-Traditional Techniques, *International Journal of Engineering Science and Technology,* 2(12), 2010, pp. 7289-7296.
14. Subbaiah, K. V., Nageswara Rao, M., and Narayana Rao, K., Scheduling of AGVs and Machines in FMS with Makespan Criteria Using Sheep Flock Heredity Algorithm, *International Journal of Physical Sciences*, 4(2), 2007, pp. 139-148.
15. Truszkowski, W., Hinchey, M., Rash, J., and Rouff, C., NASA's Swarm Missions: The Challenge of Building Autonomous Software, *IT Professional*, 06(5), 2004, pp. 47–52.
16. Ulusoy, G., Sivrikaya-Serifoglu, F., and Bilge, U., A Genetic Algorithm Approaches to the Simultaneous Scheduling of Machines and Automated Guided Vehicles, *Computer and Operations Research,* 24(4), 1997, pp. 335-351.
17. Ziebuhr, W., Ohlsen, K., Karch, H., Korhonen, T., and Hacker, J., Evolution of Bacterial Pathogenesis, *Cellular and Molecular Life Sciences,* 56, 1999, pp. 719– 728.