

**SCHEDULING OF AUTOMATED GUIDED VEHICLES IN  
A FLEXIBLE MANUFACTURING SYSTEM**

**MARYAM MOUSAVI**

**FACULTY OF ENGINEERING  
UNIVERSITY OF MALAYA  
KUALA LUMPUR**

**2018**

**SCHEDULING OF AUTOMATED GUIDED  
VEHICLES IN A FLEXIBLE MANUFACTURING  
SYSTEM**

**MARYAM MOUSAVI**

**THESIS SUBMITTED IN FULFILMENT OF THE  
REQUIREMENTS FOR THE DEGREE OF THE DOCTOR  
OF PHILOSOPHY**

**FACULTY OF ENGINEERING  
UNIVERSITY OF MALAYA  
KUALA LUMPUR**

**2018**

**UNIVERSITY OF MALAYA**  
**ORIGINAL LITERARY WORK DECLARATION**

Name of Candidate: *Marjam Mousavi* (I.C/Passport No: *H95659618* )

Matric No: *KHA120095*

Name of Degree: *phD*

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

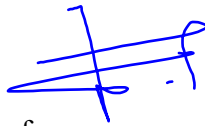
*Scheduling of automated guided vehicles in a flexible manufacturing system*

Field of Study: *Manufacturing system*

I do solemnly and sincerely declare that:

- (1) I am the sole author/writer of this Work;
- (2) This Work is original;
- (3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
- (4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
- (5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
- (6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature



Date: *12 June 2018*

Subscribed and solemnly declared before,

Witness's Signature

Date:

Name:

Designation:

# **SCHEDULING OF AUTOMATED GUIDED VEHICLES IN A FLEXIBLE MANUFACTURING SYSTEM**

## **ABSTRACT**

Flexible manufacturing systems (FMSs) provide high flexibility and responsiveness to manufacturers to meet variable customer demands in the market, where a variety of products with short production cycle is favored. Performance of an FMS is highly dependent on the superiority of the coordination and scheduling of its components like automated guided vehicles (AGVs). AGV scheduling refers to the process of allocating AGVs to tasks, taking into account the costs and time required for the operations to be accomplished. Multi-objective scheduling, in this regard, is highly complex and combinatorial in nature when conflicting objectives are involved. Minimizing makespan (the time required to complete all jobs) and the number of AGVs in an FMS would consequently minimize the production costs. In addition, AGVs' battery charge status and utilization largely affect task scheduling performance, in which without such consideration the scheduling results would be unrealistic. Incorporation of the AGVs battery charge consideration into the scheduling practice escalates the model complexity, and it has been rarely studied before. However, in practice, the AGV's battery charge status cannot be neglected. AGV scheduling is a non-deterministic polynomial-time hard (NP-hard) problem and evolutionary algorithms (EAs) have been proved powerful in solving such problems. In this study, a multi-objective optimization model for AGV scheduling in an FMS is developed and solved using four evolutionary algorithms. Genetic algorithm (GA), particle swarm optimization (PSO), and two different hybrids of GA and PSO that are referred to as HGP1 and HGP2 are the four EAs developed. In both the hybrid algorithms, to obtain an algorithm capable of finding better results with improved convergence properties, some of the GA operators such as selection, crossover, and mutation were integrated to the PSO algorithm. In HGP2, elitism integration,



application of an innovative way of population selection, and a different approach for incorporation of the GA operators into the PSO have been practiced as well. Next, the model and algorithms were applied to four testbeds in different sizes to assess the developed model and solution approaches. The four algorithms were successful in decreasing the makespan and the required number of AGVs in all the testbeds. With regard to the battery charge utilization, not only the batteries of omitted AGVs were saved, but also the remaining AGVs' battery charge utilization was improved. After the optimization, along with decrease in AGVs' number, their idle time has also been reduced and consequently the AGVs' operation efficiency was improved. Overall, in all the testbeds, HGP2 outperformed the other algorithms and obtained the best result. Moreover, HGP2 converged at a faster rate and had a smaller standard deviation and computational time. Increasing the problem size did not change the response pattern of the studied EAs, however it postponed the algorithms convergence to a higher iteration number with prolonged computational time. Finally, in order to validate the proposed model, a model simulation was performed by FlexSim software. The simulation outcome confirmed the optimization result which proved the feasibility and validity of the model.

**Keywords:** Automated guided vehicle, scheduling, optimization, flexible manufacturing system.

# ***SCHEDULING OF AUTOMATED GUIDED VEHICLES IN A FLEXIBLE MANUFACTURING SYSTEM***

## ***ABSTRAK***

Sistem pembuatan fleksibel (FMS) menyediakan daya keanjalan dan kepekaan yang tinggi bagi memenuhi permintaan pelanggan yang mendadak, di mana kepelbagaian produk dengan kitaran pengeluaran yang singkat menjadi pilihan. Prestasi sesuatu FMS adalah bergantung kepada kejituan dasar penjadualan untuk sistem kawalan. Prestasi FMS boleh ditingkatkan dengan melalui penyelarasan dan penjadualan komponennya seperti kenderaan berpandu automatik (AGV). Penjadualan AGV merujuk kepada proses penentuan tugas AGV, dengan mengambil kira kos dan masa operasi. Ini melibatkan penjadualan pelbagai objektif yang bersifat kompleks dan kombinatorik di mana ia tidak mempunyai satu penyelesaian unik yang boleh dicapai apabila ia melibatkan objektif yang bercanggah. Pengurangan bilangan AGV disamping meminimumkan masa penyiapan dalam FMS seterusnya akan mengurangkan kos pengeluaran. Selain itu, status pengecajan bateri dan penggunaan AGV memberi kesan penting pada penjadualan tugas, sekiranya tanpa pertimbangan status-status tersebut, ia akan menyebabkan keputusan penjadualan jauh berbeza daripada realiti. Penglibatan proses pengecajan bateri dalam penjadualan akan meningkatkan kerumitan model. Walaupun dalam amalan industri status pengecasan bateri AGV ini tidak boleh diabaikan, kajian berkaitan perkara ini tidak pernah dilakukan oleh penyelidik-penyelidik sebelum ini. Penjadualan AGV merupakan masalah polinomial-masa yang rumit dan algoritma evolusi telah dibuktikan sebagai alat yang berkesan untuk mengatasi masalah pengoptimuman tersebut. Dalam kajian ini, model pengoptimuman pelbagai objektif untuk penjadualan AGV dalam FMS telah dibangunkan dan diselesaikan dengan menggunakan algoritma evolusi. Empat algoritma evolusi iaitu Algoritma Genetik (GA), Pengoptimum Kerumunan Zarah (PSO), dan dua gabungan berbeza GA dan PSO yang dirujuk sebagai algoritma HGP1 dan HGP2 telah

digunakan. Dalam kedua-dua algoritma hibrid yang dibangunkan, beberapa operator GA seperti pemilihan, persilangan dan mutasi telahpun diintegrasikan dengan algoritma PSO untuk mendapatkan algoritma yang mampu mendapat hasil yang lebih baik dengan sifat penumpuan yang lebih baik. Dalam HGP2, penerapan cara pemilihan yang inovatif dan pendekatan yang berbeza untuk penubuhan operator GA ke PSO telahpun digunakan. Seterusnya, model dan algoritma telah digunakan untuk empat kajian dalam pelbagai saiz untuk menilai model yang dibangunkan dan cara-cara penyelesaian. Keempat-empat algoritma telah berjaya mengurangkan masa penyiapan dan jumlah AGV yang diperlukan dalam semua kajian. Dari segi penggunaan pengecas bateri, bukan sahaja kadar pembaziran bateri AGV yang dikurangkan, malah penggunaan pengecas bateri AGV yang tinggal juga diperbaiki. Selepas pengoptimuman, selain daripada penurunan bilangan AGV, masa terbiar AGV juga dikurangkan dan seterusnya kecekapan operasi AGV telah ditingkatkan. Secara keseluruhan, dalam semua kajian yang dijalankan, prestasi HGP2 adalah mengatasi algoritma yang lain dan mendapat keputusan yang terbaik. Selain itu, kadar penumpuan HGP2 adalah lebih cepat dan mempunyai sisihan piawai dan masa pengiraan yang lebih kecil. Peningkatan saiz masalah tidak mengubah corak tindak balas EA yang dikaji, bagaimanapun ia dapat menanggukkan penumpuan algoritma kepada nombor lelaran yang lebih tinggi dengan masa pengiraan yang berpanjangan. Akhirnya, perisian FlexSim telah digunakan untuk mengesahkan model yang dibangunkan. Hasil simulasi perisian ini adalah selari dengan keputusan yang diperolehi. Keputusan ini telah mengesahkan keputusan pengoptimuman dan sekaligus membuktikan kesahihan dan kesesuaian model yang dibangunkan.

**Keywords:** Kenderaan berpandu automatic, penjadualan, pengoptimuman, sistem perkilangan fleksibel.

## ACKNOWLEDGEMENTS

First and foremost, I would like to express my profound gratitude, especially to my supervisor Assoc. Prof. Dr. Yap Hwa Jen, for continuous support of my Ph.D study and all of the valuable guidance, suggestions, support, encouragements, understanding, and patience.

Besides, I would like to thank my co-supervisor, Dr. Siti Nurmayana Musa for her insightful and valuable comments, suggestion, and encouragement, understanding, and help.

In particular, I am grateful to Dr. Farzad Tahriri for enlightening me the first glance of this research topic and for his valuable guidance throughout the research.

I am also thankful to Assoc. Prof. Dr. Siti Zawiah for the financial support of this research project.

I would like to dedicate this thesis to:

*My dear husband, Dr. Hadi Galavi for his patience, devotion, help, and his endless support throughout writing this thesis.*

*My beloved father's memory, my dearest mother, my loving sister, and my supportive brothers who have always loved me unconditionally.*

## TABLE OF CONTENTS

Abstract .....	iv
Abstrak .....	vi
Acknowledgements.....	viii
Table of Contents.....	ix
List of Figures.....	xiii
List of Tables.....	xvi
List of Symbols and Abbreviations .....	xviii
<b>CHAPTER 1: INTRODUCTION.....</b>	<b>1</b>
1.1 Introduction.....	1
1.2 Problem Statement.....	4
1.3 Objectives.....	5
1.4 Significance of the Study.....	5
1.5 Scope of the Research.....	6
1.6 Thesis layout.....	6
<b>CHAPTER 2: LITERATURE REVIEW.....</b>	<b>7</b>
2.1 Introduction.....	7
2.2 Flexible Manufacturing System (FMS).....	7
2.2.1 FMS Components .....	7
2.2.2 Benefits of FMS.....	8
2.3 Automated Guided Vehicle System.....	9
2.3.1 Different Types of AGVs .....	10
2.3.2 AGV Guidance System .....	10
2.3.3 Guide Path in AGVS.....	11

2.4	AGV Scheduling.....	11
2.4.1	On-line vs. Off-line Scheduling.....	13
2.4.2	Methods of AGV Scheduling .....	14
2.5	Evolutionary Algorithm .....	17
2.5.1	Genetic Algorithm .....	18
2.5.2	Particle Swarm Optimization.....	20
2.5.3	Hybrid Algorithms .....	22
2.5.4	Multi-objective optimization .....	24
2.6	The current research establishment .....	25
2.7	Summary .....	26
<b>CHAPTER 3: METHODOLOGY .....</b>		<b>28</b>
3.1	Introduction .....	28
3.2	Research Framework .....	28
3.3	Model Derivation.....	29
3.3.1	Multi-objective Evaluation.....	36
3.3.1.1	Minimizing the Makespan .....	37
3.3.1.2	Minimizing the Number of AGVs.....	39
3.4	Optimization Algorithms Developed for the Model.....	42
3.4.1	Genetic Algorithm .....	43
3.4.2	Particle Swarm Optimization.....	50
3.4.3	Hybrid GA and PSO .....	56
3.4.3.1	The First Hybrid GA-PSO (HGP1).....	56
3.4.3.2	The Second Hybrid GA-PSO (HGP2).....	57
3.5	Programming in MATLAB.....	60
3.6	Evaluation of Model and Algorithms .....	61
3.7	Validation .....	62

3.7.1	Model Validation .....	62
3.7.2	Validation of Optimization Results .....	63
3.8	Summary .....	64
<b>CHAPTER 4: RESULTS AND DISCUSSION .....</b>		<b>65</b>
4.1	Introduction .....	65
4.2	The Developed Model.....	65
4.3	Evolutionary Algorithms.....	66
4.4	Model and Algorithms' Performance Evaluation.....	66
4.4.1	Parameter Setting of the Algorithms.....	66
4.4.2	Performance at Testbed 1 .....	67
4.4.2.1	Makespan and Number of AGVs .....	68
4.4.2.2	AGVs' Battery Charge .....	72
4.4.2.3	AGVs' Specifications/Behavior.....	73
4.4.3	Performance at Testbed 2 .....	76
4.4.3.1	Makespan and Number of AGVs .....	77
4.4.3.2	AGVs' Battery Charge .....	81
4.4.3.3	AGVs' Specifications/Behavior.....	82
4.4.4	Performance at Testbed 3 .....	84
4.4.4.1	Makespan and Number of AGVs .....	86
4.4.4.2	AGVs' Battery Charge .....	90
4.4.4.3	AGVs' Specifications/Behavior.....	91
4.4.5	Performance at Testbed 4 .....	94
4.4.5.1	Makespan and Number of AGVs .....	96
4.4.5.2	AGVs' Battery Charge .....	101
4.4.5.3	AGVs' Specifications/Behavior.....	102
4.4.6	Testbed-size Effect on Model and EAs.....	106

4.4.7	EAs inter-comparison.....	108
4.5	Validation of the Optimization Model.....	110
4.6	Validation of Optimization Result.....	114
4.6.1	Layout Set up.....	114
4.6.2	Model’s Rules and Information Entry to the FlexSim Database.....	115
4.6.3	Simulation Result.....	118
4.7	Summary.....	120
 <b>CHAPTER 5: CONCLUSIONS .....</b>		<b>122</b>
5.1	Research Summary.....	122
5.2	Conclusions.....	123
5.3	Future Research.....	125
References .....		127
List of Publications and Papers Presented .....		142
Appendix: Model and Algorithms Programming.....		143



## LIST OF FIGURES

Figure 2.1: AGV components.....	10
Figure 2.2: Different types of AGVs .....	10
Figure 3.1: The overall research framework .....	29
Figure 3.2: General flowchart of the multi-objective optimization model .....	31
Figure 3.3: The detailed flowchart of the multi-objective optimization model .....	32
Figure 3.4: Pseudocode of the model.....	33
Figure 3.5: Flowchart of the GA .....	43
Figure 3.6: Example of one-point crossover (Mousavi et al., 2017) .....	47
Figure 3.7: Example of two-point crossover.....	47
Figure 3.8: Repairing offsprings out of one-point crossover (Mousavi et al., 2017) .....	48
Figure 3.9: An example of repairing offsprings out of two-point crossover .....	48
Figure 3.10: Example of shift mutation operator (Mousavi et al., 2017) .....	49
Figure 3.11: Flowchart of PSO.....	51
Figure 3.12: Flowchart of HGP1 .....	57
Figure 3.13: Flowchart of HGP2 .....	58
Figure 4.1: The layout of testbed 1 .....	68
Figure 4.2: Performance of the four algorithms at testbed 1.....	69
Figure 4.3: Best performance (minimum) of the four algorithms at testbed 1.....	70
Figure 4.4: Operations' sequence before optimization – testbed 1 .....	72
Figure 4.5: Operations' sequence after optimization by HGP2 – testbed 1.....	72
Figure 4.6: AGVs' battery charge consumption, before and after optimization .....	73
Figure 4.7: Battery charge utilization, before and after optimization – testbed 1 .....	73
Figure 4.8: AGVs' specification before optimization .....	74

Figure 4.9: AGVs' specification after optimization .....	74
Figure 4.10: AGVs' operation efficiency before and after optimization.....	75
Figure 4.11: The layout of testbed 2 .....	76
Figure 4.12: Performance of four algorithms at testbed 2 .....	77
Figure 4.13: Best performance (minimum) of the four algorithms at testbed 2.....	78
Figure 4.14: Operations' sequence before optimization – testbed 2 .....	80
Figure 4.15: Operations' sequence after optimization by HGP2 – testbed 2.....	80
Figure 4.16: AGVs' battery charge consumption, before and after optimization .....	81
Figure 4.17: Battery charge utilization, before and after optimization – testbed 2 .....	82
Figure 4.18: AGVs' specification before optimization.....	82
Figure 4.19: AGVs' specification after optimization .....	83
Figure 4.20: AGVs' operation efficiency before and after optimization.....	84
Figure 4.21: The layout of testbed 3 .....	85
Figure 4.22: Performance of the four algorithms at testbed 3 .....	86
Figure 4.23: Best performance of the four algorithms at testbed 3 .....	87
Figure 4.24: Operations' sequence before optimization – testbed 3 .....	89
Figure 4.25: Operations' sequence after optimization by HGP2 – testbed 3 .....	89
Figure 4.26: AGVs' battery charge consumption, before and after optimization .....	90
Figure 4.27: Battery charge utilization, before and after optimization – testbed 3 .....	91
Figure 4.28: AGVs' specification before optimization.....	92
Figure 4.29: AGVs' specification after optimization .....	93
Figure 4.30: AGVs' operation efficiency before and after optimization.....	94
Figure 4.31: The layout of testbed 4 .....	95
Figure 4.32: Performance of four algorithms at testbed 4 .....	97

Figure 4.33: Best performance (minimum) of the four algorithms at testbed 4.....	97
Figure 4.34: Operations' sequence before optimization – testbed 4 .....	100
Figure 4.35: Operations' sequence after optimization by HGP2 – testbed 4.....	100
Figure 4.36: AGVs' battery charge consumption before and after optimization.....	101
Figure 4.37: Battery charge utilization, before and after optimization – testbed 4.....	102
Figure 4.38: AGVs specification before optimization.....	103
Figure 4.39: AGVs' specification after optimization .....	104
Figure 4.40: AGVs' operation efficiency before and after optimization.....	105
Figure 4.41: Best performance of the four algorithms at four testbeds .....	106
Figure 4.42: Layout 1 (Bilge & Ulusoy, 1995) .....	110
Figure 4.43: layout 2 (Bilge & Ulusoy, 1995) .....	111
Figure 4.44: Sink, source, AGV, and machines in the simulated model environment.	115
Figure 4.45: Part of the “job table” in FlexSim.....	115
Figure 4.46: Source properties .....	116
Figure 4.47: Table of operation time of the example in FlexSim .....	117
Figure 4.48: Machine properties.....	117
Figure 4.49: Completion time for each job .....	118
Figure 4.50: Waiting time, processing time and travelling time of the goods .....	119
Figure 4.51: Buffering queues' time on collecting, releasing and being empty .....	119
Figure 4.52: Simulation environment, when the model is running .....	120

## LIST OF TABLES

Table 2.1: AGV scheduling literature (objectives and methods) .....	16
Table 2.2: Studies with similar research components.....	25
Table 3.1: General schematic for reading data (Mousavi et al., 2017).....	44
Table 3.2: Encoding of a sample particle (Mousavi et al., 2017).....	53
Table 4.1: Different settings of parameters experimented .....	67
Table 4.2: AGV travel time (minutes) among L/U point and machines.....	68
Table 4.3: The processing time (minutes) of every operation on the machines.....	68
Table 4.4: Test results of optimization algorithms at testbed 1 for hundred runs .....	71
Table 4.5: AGV travel time (minutes) between L/U points and machines.....	76
Table 4.6: The processing time (minutes) of every operation on different machines ....	76
Table 4.7: Test results of optimization algorithms at testbed 2 for hundred runs .....	79
Table 4.8: AGV travel time (minutes) among L/U point and machines.....	85
Table 4.9: The processing time (minutes) of every operation on the machines.....	85
Table 4.10: Results of optimization algorithms at testbed 3 for hundred runs .....	88
Table 4.11: AGV travel time (minutes) among L/U point and machines.....	95
Table 4.12: The processing time (minutes) of every operation on the machines.....	96
Table 4.13: Test results of optimization algorithms at testbed 4 for hundred runs .....	98
Table 4.14: Test results of optimization algorithms at four testbeds for hundred runs	107
Table 4.15: Travel time (minutes) among L/U and machines – layout 1 (Bilge & Ulusoy, 1995).....	111
Table 4.16: Travel time (minutes) among L/U and machines – layout 2 (Bilge & Ulusoy, 1995).....	111
Table 4.17: Processing time (minutes) of operations on the machines (Bilge & Ulusoy, 1995).....	112

Table 4.18: Comparison of makespan results ..... 113

## LIST OF SYMBOLS AND ABBREVIATIONS

$n$	: Total number of jobs	
$j, j'$	: Indexes of jobs, genes	code, $j, j' = 1, 2, \dots, n$
$m_j$	: Total number of operations for each job $j$	
$i, i'$	: Indexes of operations, $i, i' = 1, 2, \dots, m_{j, j'}$	
$\theta$	: Total number of operations for all of jobs	
$z$	: Number of AGVs	
$a, a'$	: Index of AGVs, $a, a' = 1, \dots, z$	
$y$	: Index of new AGV	
$J_j$	: Job number $j$	
$O_{ji}$	: Operation $i$ of job $j$ , $O_{j\_i}$ for $j \geq 10 \wedge / \vee i \geq 10$	
$M_{ji}$	: Assigned machine for $O_{ji}$	
$p_{ji}$	: Processing time of $O_{ji}$	
$p_{ji}^s$	: Start time of processing $O_{ji}$	
$p_{ji}^e$	: End time of processing $O_{ji}$	
$H$	: Loading/unloading point (Home)	
$A^a$	: AGV number $a$	
$T_{ji}$	: Related task to $O_{ji}$ (Moving from $M_{j(i-1)}$ to $M_{ji}$ or $H$ to $M_{ji}$ )	
$T_{ji}^a$	: Assigned $A^a$ to do task $T_{ji}$	
$T^a$	: A collection of operations that have done by $A^a$	
$T$	: A collection of $T^a$	
$MS$	: Makespan	

$PS$	: Population size for GA
$r$	: Index of chromosomes, $r = 1, \dots, PS$
$e$	: Index of genes, $e = 1, \dots, \theta$
$C_r$	: Chromosome
$G_e$	: Gene
$CR$	: Crossover rate
$Pm$	: Mutation rate
$G_{\max}$	: Maximum gene code
$It_{\max}$	: The maximum iterations
$It$	: The current iteration number
$t$	: Iteration number
$S^t$	: Swarm size at iteration ( $t$ )
$\alpha$	: Index of particles, $\alpha = 1, \dots, S^t$
$PR_\alpha$	: Particle
$d$	: Dimension, $d = 1, \dots, \theta$
$\omega$	: Inertia factor
$\omega_{\max}$	: Maximum inertia factor
$\omega_{\min}$	: Minimum inertia factor
$v_{ad}^t$	: The velocity of $\alpha^{th}$ particle on $d^{th}$ dimension at iteration ( $t$ )
$v_{ad}^{t+1}$	: The velocity of $\alpha^{th}$ particle on $d^{th}$ dimension at iteration ( $t+1$ )
$V_\alpha^t$	: The velocity of $\alpha^{th}$ particle in the swarm at iteration ( $t$ )
$q_{\alpha d}^t$	: The position of $\alpha^{th}$ particle on $d^{th}$ dimension at iteration ( $t$ )

- $q_{\alpha d}^{t+1}$  : The position of  $\alpha^{th}$  particle on  $d^{th}$  dimension at iteration  $(t+1)$
- $Q_{\alpha}$  : The position of  $\alpha^{th}$  particle in the swarm at iteration  $(t)$
- $B_{\alpha d}^t$  : The best position of  $\alpha^{th}$  particle on  $d^{th}$  dimension found so far
- $G_d^t$  : The global best position of the swarm on  $d^{th}$  dimension found so far
- $\varphi_1$  and  $\varphi_2$  : Uniformly distributed random numbers in the interval  $[0, 1]$ .
- $C_1$  : Self-confidence
- $C_2$  : Swarm confidence
- $NA$  : Number of AGVs to do all the operations
- $CIO_{ji}$  : The time that operation  $O_{ji}$  completes
- $CC h A^a$  : Current battery charge of  $A^a$
- $ChHI_{ji}^a$  : Charge that  $A^a$  needed for doing the task  $T_{ji}$  and return home
- $ChT_{ji}^a$  : The battery charge that  $A^a$  consumes for doing  $T_{ji}$
- $ChA^a$  : The total battery charge that  $A^a$  consumes for all of its operations
- $CA^a$  : Current position of  $A^a$ , (Can be  $H$ ,  $M_{ji}$ ,  $M_{j'}$ , and  $M_{j(i-1)}$ )
- $tCA^a$  : Time of current position of  $A^a$
- $tT_{ji}^a H$  : Time that  $A^a$  arrives home after doing  $T_{ji}$
- $PT_{ji}^a$  : Pick-up point of  $A^a$  doing  $T_{ji}$ , ( $P$  represents pick-up point and can be  $H$ ,  $M_{ji}$ ,  $M_{j'}$ , and  $M_{j(i-1)}$ )
- $tPT_{ji}^a$  : Pick-up time of  $A^a$  doing  $T_{ji}$
- $rPT_{ji}^a$  : The time that  $A^a$  reaches pick-up place of  $T_{ji}$



- Drop-off point of  $A^a$  doing  $T_{ji}$ , ( $D$  represents drop-off point and can be
- $DT_{ji}^a$  :  $H, M_{ji}, M_{j'j'}$ , and  $M_{j(i-1)}$ )
- $tDT_{ji}^a$  : Drop-off time of  $A^a$  doing  $T_{ji}$
- $rDT_{ji}^a$  : The time that  $A^a$  reaches drop-off place of  $T_{ji}$
- $\mu$  : A large positive number
- $tCPT_{ji}^a$  : The travel time of  $A^a$  from its current point to reach the start point of  $T_{ji}$
- $\gamma$  : A coefficient for transforming energy consumption to time
- $UT_{ji}^a$  : Unloaded time of  $A^a$  doing  $T_{ji}$
- $UtA^a$  : Total unloaded time of  $A^a$
- $ItA^a$  : Total idle time of  $A^a$
- $\overline{TuA^a}$  : The time that  $A^a$  is being charged
- $WT_{ji}^a$  : Waiting time of  $A^a$  doing  $T_{ji}$
- $WtA^a$  : Total waiting time of  $A^a$
- $LT_{ji}^a$  : loaded time of  $A^a$  doing  $T_{ji}$
- $LtA^a$  : Total loaded time of  $A^a$
- $RT_{ji}^a$  : Running time (loaded + unloaded) of  $A^a$  doing  $T_{ji}$
- $RtA^a$  : Total running time (loaded + unloaded) of  $A^a$
- $BU^a$  : Consumed battery charge utilization of  $A^a$
- $EA^a$  : Operation efficiency of  $A^a$
- $\lambda$  : A coefficient for determining when a new AGV should be added
- $L$  : Number of objectives
- $\beta$  : Index of  $\delta$ ,  $\beta = 1, \dots, L$

$\delta$	: The $\beta^{th}$ weight of the $\beta^{th}$ objective function
$\psi$	: A ratio to make balance among objectives with different ranges of value
$f(x)$	: Fitness function
FMS	: Flexible manufacturing system
AGVS	: Automated guided vehicle system
EA	: Evolutionary algorithm
GA	: Genetic algorithm
PSO	: Particle swarm optimization
NP-hard	: Non-deterministic polynomial-time hard
NC	: Numerical control
AMHS	: Automated material handling system
AS/RS	: Automated storage/retrieval systems
TAGV	: Tandem AGV
P/D	: Pick-up/drop-off
L/U	: Loading/unloading
CFRP	: Conflict-free routing problem
ACA	: Ant colony algorithm
VRP	: Vehicle routing problem
BJS	: Blocking job shop
MAS	: Multi agent-based system
SA	: Simulated annealing
CMS	: Cellular manufacturing systems
NFL	: No free lunch
SPV	: Smallest position value
HGP1	: First hybrid GA-PSO
HGP2	: Second hybrid GA-PSO

LDIW : linearly decreasing inertia weight  
OpenGL : Open graphics library  
GPU : Graphics processing unit  
FSP : FlexSim software products, Inc.  
3D : Three-dimensional

## CHAPTER 1: INTRODUCTION

### 1.1 Introduction

In today's competitive market, customer satisfaction is an important challenge to consider. Therefore, organizations have shifted their concentration from producing large quantities of a single product to a variety of products, improving their quality and timely delivery to respond to the variable customer demand. Flexible manufacturing system (FMS) is an agile system with wide flexibility which is well suited for simultaneous production of an extensive variety of parts in low volumes. FMS is a complex system consisting of elements like workstations, automated storage/retrieval systems, and material handling devices such as robots and automated guided vehicles (AGVs). AGVs are widely used in FMS due to their flexibility and compatibility in/to the system (Blazewicz et al., 1991; Reddy & Rao, 2011).

Industry 4.0 or the fourth industrial revolution is the current trend of automation and data exchange in manufacturing technologies and it is about to change the way of producing and transferring products and parts in warehouses and factory layouts. In industry 4.0, it is explained that systems would digitally be connected to machines creating flexibility and predictability in companies to stay competitive in the market (Lasi et al., 2014; S. Wang et al., 2016). However, automation is a broad area and there are many ways to reduce manual work in factories and warehouses. Introducing AGVs usually gives an appealing combination of high flexibility and low installation cost, and it is one quick way to start this revolution in companies. These systems have been practiced for decades and are today well established in many types of applications (Almada-Lobo, 2016; Rüßmann et al., 2015).

Due to the AGVs wide range of applicability, a drastic increase in AGVs global market value from US\$ 838.3 million in 2015 to US\$ 2.3 billion at the end of 2024 has been predicted (Bioportfolio, 2017).

FMS performance can be improved by effective utilization of its resources and better coordination and scheduling of its components like AGV (Fauadi & Murata, 2010; Kumar et al., 2011; Pan et al., 2013; Udhayakumar & Kumanan, 2010; Zheng et al., 2013). The term ‘scheduling’ refers to the process of allocating AGVs to tasks, taking into account the costs and time required for the operations to be done (Udhayakumar & Kumanan, 2010). Efficient scheduling therefore would increase the productivity and reduce the cost while the entire fleet is optimally utilized (Fauadi & Murata, 2010).

In view of the vast variety of objectives, limitations and considerations in scheduling context, it is still an open area of research to improve it for real-environment results. Literature has shown a great tendency toward multi-objective scheduling of AGV systems and FMSs, in which the makespan minimization criterion is accompanied with several other criteria to entertain an actual-practice scheduling (Fazlollahtabar & Shafieian, 2014; Kato & Shin, 2010; Novas & Henning, 2014). The term “Makespan” refers to the completion time of all jobs in the schedule. In the majority of earlier studies, makespan minimization was the main objective in the scheduling practice as it reduces the time of production and warehousing and leads to overall cost reduction (Huang & Zhang, 2013a; Saidi-Mehrabad et al., 2015; Zheng et al., 2013). However, those studies have discounted the importance of proper utilization of FMS components. Minimizing makespan without considering the total number of AGVs employed, may increase the production costs through unnecessary utilization of a large number of AGVs in the FMS. Allocating a large number of AGVs shortens the makespan, which seems to reduce the costs at the first sight, but it will mount the idle time of AGVs and pertinent expenses (Azimi, 2011).

AGVs are such expensive devices that determining the type and the appropriate number of them in an FMS can positively influence the profitability of the business (Aized, 2009; Kato & Shin, 2010; Liang et al., 2012; Wang & Chan, 2014; Wang et al., 2014).

Another challenge in AGV scheduling studies is the inclusion of AGV battery charge considerations into the model. Many studies make the assumption of having an AGV with full battery charge at all time in their scheduling, which leaves the model impractical (Oliveira et al., 2012; Vivaldini et al., 2013). Battery management is crucial to the AGV System (AGVS) efficiency as it can reduce the costs and increase the productivity of the FMS (Kawakami & Takata, 2012; Oliveira et al., 2011). Inclusion of the AGV battery charge considerations into the scheduling practice would enhance the practicality and competency of the scheduled system.

AGV scheduling is a non-deterministic polynomial-time hard (NP-hard) problem, in which it requires application of metaheuristic methods like evolutionary algorithms (EA) to solve it. EAs are well received by the research community because of their ability to tackle problems that are highly complex. Genetic algorithm (GA) and particle swarm optimization (PSO) are two of the well-known EAs in scheduling discourse. In previous studies, GA has been more extensively used in AGV scheduling compared with other algorithms and hybrids. However, application wise, every algorithm can be a suitable choice for problems of a certain type only (Wolpert & Macready, 1997). Performance of EAs can be improved by the proper choice of their operators and parameters. In addition, hybridization of these algorithms may also further improve their performances.

To address the above concerns, this research aimed to schedule AGVs in an FMS environment by developing a multi-objective model that minimizes the makespan and total number of employed AGVs while considering the AGVs' battery charge status. The model will be optimized using four evolutionary algorithms (genetic algorithm (GA),

particle swarm optimization (PSO), and 2 different hybrids of GA and PSO (so called HGP1 and HGP2)). It will be validated through testbeds run and simulation in FlexSim software.

## **1.2 Problem Statement**

Efficient scheduling would improve the system productivity and reduce the costs while the entire AGV fleet is optimally utilized (Fauadi & Murata, 2010). Previous studies have shown that multi-objective models produce a better result than single-objective ones in AGV scheduling. Having a wide variety of scheduling criteria, it is difficult to integrate all the criteria in one model. Therefore, each study optimizes the scheduling for a few of the objectives. The exhaustive literature review in this study revealed that the potential of AGV scheduling with objective setting of minimizing the number of AGVs and makespan while considering the AGVs' battery charge has not been studied yet.

Makespan minimization reduces the time of production and warehousing and saves cost (Saidi-Mehrabad et al., 2015). Next, the number of AGVs employed heavily influences the performance of an AGV system and the production cost-effectiveness as they are expensive devices (Liang et al., 2012). In addition, it is necessary to take into consideration that the appropriate use of AGVs' battery can affect the overall performance of the AGV system (AGVS) through saving cost and avoiding battery-oriented interruptions and deadlocks (Kawakami & Takata, 2012). Therefore, integrating the above criteria in a scheduling model, can result in saving time, energy, and cost.

To find the efficient solution approach for the scheduling problem, literature introduces the evolutionary algorithms as an appropriate choice for solving NP-hard problems such as scheduling (Hurink & Knust, 2005). Every EA can be suitable for a certain type of problem ; GA and PSO are two of the highly cited algorithms for solving scheduling problems (Zhang et al., 2011); however, the hybrid of GA and PSO is commonly believed

to be more effective than its constituent algorithms (Mehta, 2012). Although literature has practiced hybridization of the GA and PSO, due to the many possibilities in the choice of operators and parameters integration strategy in the hybrid form, it is always novel to find a better strategy for obtaining the optimum result in a specific problem. Thus, hybridization of two well-known algorithms of GA and PSO through a new integration approach (called HGP2) is accomplished for model optimization in this study. However, GA, PSO, and another hybrid of GA and PSO (HGP1) were also developed and compared with HGP2.

### **1.3 Objectives**

According to the research opportunities discussed above, the main objective in this study is to develop a multi-objective optimization model for scheduling of AGVs in an FMS. To achieve this aim, the following objectives are delineated:

1. To develop an AGV scheduling optimization model with multiple objectives/criteria for an FMS environment.
2. To develop evolutionary algorithms for AGV scheduling optimization model.
3. To validate optimized result by discrete event simulation.

### **1.4 Significance of the Study**

Productivity of an FMS is highly dependent upon its components scheduling and fast synchronization to the system interventions and/or interruptions. AGVs with a fast-growing global market—especially in Asia Pacific—are one expensive and widely used component of the FMSs that their scheduling greatly impacts the FMS productivity and profitability. The type of criteria/objectives integrated together to develop a scheduling model can affect the system responsiveness. This study postulates that a multi-objective scheduling model with the following criteria can provide a seamless scheduling model with an economical utilization of resources that assures the system profitability. The



criteria are minimization of the makespan and the number of AGVs utilized in the FMS while considering their battery charge at all time. This study for the optimization of the developed scheduling model employs four EAs, which two of them are different hybrids of the GA and PSO algorithms. A novel configuration for the integration of the GA and PSO elements is used to develop the hybrid GA-PSO algorithm. Overall, the knowledge acquired from such a comprehensive approach is beneficial to both academia and engineers who aim to gain a better perspective of the AGV scheduling context.

### **1.5 Scope of the Research**

The scope of this research is developing a general model for an FMS environment. This study addresses the general scheduling problem of multiple unit-load AGVs in a plant with multiple machines arranged in a distributed layout and set of jobs to be processed and various types of products to be produced. The machine-to-machine distance and the distance between loading/unloading machines are presumed known.

### **1.6 Thesis layout**

The research fundamentals of this work were established in chapter one. Readers would find a literature review on FMS, automated guided vehicle, scheduling, and evolutionary algorithms in chapter two. Chapter three of the dissertation discusses the research framework, model derivation, optimization algorithms development, and programming in MATLAB. Results of the model and algorithms' performance, testbeds implementation, model validation, and many more are presented in chapter four. The last chapter would represent a research summary and the conclusions drawn from this project, while possible future works are also put forward for the respected readers.

## **CHAPTER 2: LITERATURE REVIEW**

### **2.1 Introduction**

According to the research objectives defined, flexible manufacturing system and its components, AGV system and its components, scheduling, AGV scheduling and its methods, and evolutionary algorithms are explained in this chapter. The literature on above topics is reviewed and the prominent studies on AGV scheduling are summarized. Next, evolutionary algorithms have been studied and described with the research focus being set on genetic algorithm and particle swarm optimization and their hybrid. The literature reviewed here are the basis for constructing the research methodology and overall framework of the study.

### **2.2 Flexible Manufacturing System (FMS)**

A flexible manufacturing system is a “reprogrammable” manufacturing system capable of producing a variety of products automatically. The various machining cells are interconnected via loading and unloading stations and through an automated transport system. Operational flexibility is enhanced by the ability to execute all manufacturing tasks on numerous product designs in small quantities with fast delivery. It has been described as an automated job shop and as a miniature-automated factory. Simply stated, an automated production system produces one or more families of parts in a flexible manner. Today, this prospect of automation and flexibility presents the possibility of producing nonstandard parts to create a competitive advantage. The general objectives of an FMS are to approach the efficiencies and economies of a scale normally associated with mass production, and to maintain the flexibility required for small- and medium-lot-size production of a variety of parts (Chandraa et al., 2015; Srivastava et al., 2008).

#### **2.2.1 FMS Components**

A generic FMS consists of the following components:

- Numerical control (NC) machine tools. A set of work stations containing machine tools that do not require significant set-up time or changeover between successive jobs. Typically, these machines perform milling, boring, drilling, tapping, reaming, turning, and grooving operations (Kumar et al., 2006).
  
- Automated material handling system (AMHS). A material-handling system is automated and flexible in which it permits jobs to move between any pair of machines so that any job routing can be followed (Chandreaa et al., 2015). AMHS can be divided in three groups as follows:
  - Automated guided vehicles
  - Conveyors
  - Automated storage and retrieval systems (AS/RS)
  
- Industrial robots. Industrial robots minimize the role of human labor, allowing rapid changes to assembly lines, avoiding costly equipment replacements, and enabling the economical production of customized lots (Sciavicco & Siciliano, 1996).
  
- Control software. Control software is a network of supervisory computers and microprocessors that performs some or all of the following tasks: (a) directs the routing of jobs through the system; (b) tracks the status of all jobs in progress so it is known where each job is to go next; (c) passes the instructions for the processing of each operation to each station and ensures that the right tools are available for the job; and (d) provides essential monitoring of the correct performance of operations and signals problems requiring attention (Ficko et al., 2004; Oyetunji, 2012).

### **2.2.2 Benefits of FMS**

Numerous researchers have detailed the potential benefits of FMS implementation. These benefits include: less waste, fewer workstations, “quicker changes of tools, dies, and

stamping machinery”, reduced downtime, better control over quality, reduced labor, more efficient use of machinery, work-in-process inventory reduced, increased capacity, increased production flexibility (Haq et al., 2003; Karsak & Kuzgunkaya, 2002; Malhotra et al., 2010; Pandey et al., 2016; Tseng, 2004).

### **2.3 Automated Guided Vehicle System**

AGVs are one of the commonly favored types of vehicles for the transfer of raw material, working process, finish parts, tools, and supplies among different points, machines, and the components of the manufacturing system in an economic way in FMSs. AGVs were introduced in 1955 (Muller, 1983). Since then, AGVs’ applications and types have significantly evolved. AGVs need a close monitoring and effective control strategies because of their automated system (Albert & Castagna, 1996; Martínez-Barberá & Herrero-Pérez, 2010). They are cordless and their program can change based on the path designs; thus, they increase the flexibility for flow changes within a facility. As automation and flexibility have become crucial factors in material handling, AGVs are found perfect for low and medium -volume material handling situations, where the routing of materials is more individualized (Albert & Castagna, 1996; Hall et al., 2001; Ilić, 1994).

A number of AGVs working together in a facility constitute an AGV system (AGVS). An AGVS is comprised of four main components; (1) the vehicles that are unmanned devices for material transportation within the system, (2) the guide path that guides the vehicle to move along the path, (3) the control unit which observes the system and guides the operations, moves, etc., (4) and the computer interface which connects the AGVS with other computers and systems such as mainframe host computer and the FMS (Figure 2.1).

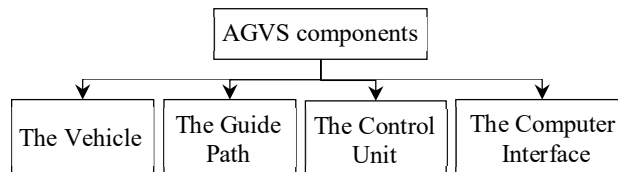


Figure 2.1: AGV components

### 2.3.1 Different Types of AGVs

Various AGVs that accommodate different service requirements are shown in Figure 2.2. However, AGVs with trailers (Tow/Tugger) designed for material transport between workstations within an FMS are the common vehicle types in manufacturing industry.

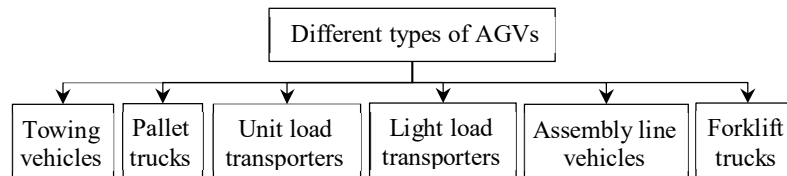


Figure 2.2: Different types of AGVs

### 2.3.2 AGV Guidance System

The modern AGVs are free-ranging vehicles that are available in limited types with higher cost. Their preferred tracks are computer-programmed and uploaded to the vehicles' controllers, and they are changeable. These vehicles find their way using odometer, gyroscope, laser, magnetic, vision, or radiofrequency techniques (Le-Anh & De Koster, 2006; Tompkins et al., 2010). Having no operator, AGVs follow a set of guide paths in the facility layout synchronized using a computer-based control system. The guidance system assures the AGVs movement on the track/predefined path. AGV's type, application, requirements, and imposed environmental limitations define the type of guidance systems to be employed. The wire, laser, inertial, optical or painted strip, infrared, and teaching-type guidance systems are the well-known systems.

### **2.3.3 Guide Path in AGVS**

The guide paths in AGVS are categorized into tandem, single-loop, and conventional ones. By configuration, the tandem AGV (TAGV) system classifies the workstations into non-overlapping zones and assigns one AGV exactly to every zone. Scheduling problems are quite simple in a tandem system, while they are highly complicated in conventional systems. A conventional system corresponds to sophisticated network with crosses, paths, junctions, and shortcuts. AGVs may travel through a path in a single direction (i.e., unidirectional path) or in both directions (i.e., bidirectional path) (Le-Anh & De Koster, 2006). With respect to the vehicles use and probable throughput efficiency, it is argued that the bidirectional path is more advantageous than the unidirectional path systems (Qiu et al., 2002).

## **2.4 AGV Scheduling**

Scheduling is the process of generating the schedule. Scheduling problems in industry contain a set of tasks to be carried out and a set of resources available to perform those tasks. Given tasks and resources, together with some information about uncertainties, the general problem is to determine the timing of the tasks while recognizing the capability of the resources. In the scheduling process, it is needed to know the type and the amount of each resource to determine when the tasks can feasibly be accomplished. In fact, information about resources and tasks defines a scheduling problem (Baker, 1995; Pinedo, 2016).

AGV scheduling is one of the major aspects of AGVs application and management. The term ‘AGV scheduling’ refers to the process of allocating AGVs to tasks taking into account the costs and time of operations and warranting conflict-free paths (Udhayakumar & Kumanan, 2010). The goal of AGV scheduling is to release a group of AGVs in order to accomplish the objectives for a cluster of pick-up/drop-off (P/D) tasks

under specific restrictions like priority and deadlines. The scheduling objectives are typically related to the tasks' processing time or resources utilization (the system throughput, the overall travel time of vehicles, and the number of AGVs) under specific limitations like priority and time limit. However, non-feasible outcomes can be obtained if the functioning transport system does not take into consideration the scheduling constraints (Akturk & Yilmaz, 1996; Vis, 2006). Le-Anh (2005) highlighted that the principal goal of the majority of the scheduling problems is to move loads (pallets, containers, and products) as quickly as possible to fulfill the time window restrictions. Other objectives may include minimization of the load waiting time and maximum number of items in the critical queues. However, minimization of the mean waiting time of load in AGV scheduling is also pronounced as an important objective (Le-Anh & De Koster, 2006). The AGVs' empty travel time is not the main concern of the majority of the AGV scheduling problems, as it does not preclude the transport orders (loads) in the AGVS. On the other hand, Mallikarjuna (2014) claimed that the scheduling main purpose is dependent upon the market demand, the situation, customer's satisfaction standards, and company's demands. Overall, in this context, there are two major scheduling goals:

(1) Minimizing makespan

This broad goal includes the following objectives: (i) minimizing machine's idle time, (ii) minimizing the costs of in-process inventories, (c) finishing each job the soonest possible, and (iv) finishing the last job the soonest possible.

(2) Due date-based cost minimization

The objectives involved in this goal are: (i) minimizing the costs associated with failure in meeting the scheduled date, (ii) minimizing the maximum possible delay of any job, (iii) minimizing the overall tardiness, and (iv) minimizing the number of tardy jobs.

In other respects, Akturk and Yilmaz (1996) fragmented the scheduling into two key mechanisms: (i) predictive mechanisms that specify the prescribed starting and completion times of labor operations and (ii) reactive mechanisms that monitor the progress of the schedule and handle the unexpected events (e.g., breakdowns, failures, date changes, and cancellations).

#### **2.4.1 On-line vs. Off-line Scheduling**

In off-line scheduling, all the available tasks are scheduled at the same time. Hence, with any alteration in the tasks, the previously generated schedule must be reviewed and updated all over the production cycle. Off-line scheduling denotes the scheduling of all operations of the available jobs for the whole scheduling period. If all the tasks are known when planning, then the scheduling problem may be resolved off-line.

Accordingly, on-line scheduling systems are required in order to control the vehicles. In the scheduling problems, the input data domain encircles the load arrival data (time windows and dispatched and delivered sites), distance matrix of all sites, some optional information (e.g., a parking policy), and vehicle data (e.g., vehicle speed, capacity, and type) (Sabuncuoglu & Bayız, 2000). Hence, the scheduling program may automatically and efficiently control any unpredicted event in the system.

If an off-line method is employed the process is re-programmed, while in on-line methods the decision on task scheduling is taken when some changes happen in the system. In off-line scheduling, transportation orders are known beforehand and the routes are constructed and optimized before being used by vehicles. However, any slight modifications to the time of job arrival or time of driving (jamming), or vehicle failure can affect and may damage the entire schedule (Le-Anh & De Koster, 2006). In practice, the working environments are often stochastic because the AGVs' travel time, job arrival, and loading/unloading (L/U) time may vary unexpectedly, and vehicles may crash.



Therefore, the schedule should be dynamically modified in time. The schedules ought to be adjusted when any new information on the transportation orders is received (Le-Anh & De Koster, 2006).

To recap, in off-line scheduling decisions are made based on the compile-time, in which the required information is provided. An off-line scheduling algorithm can optimally arrange the sequences in advance as it only follows a predefined plan. However, in on-line scheduling there is no prior plan to arrange the sequences accordingly and it would be a great disadvantage to on-line algorithms by representing uncertainty in sequences arrangement (Shabtay et al., 2013). Thus, due to lack of information in on-line scheduling only simple scheduling techniques could be used that often poorly perform against their off-line counterparts (Gorcitz et al., 2015; Pinedo, 2012). The present study is also organized to develop an off-line scheduling with specifications discussed before.

#### **2.4.2 Methods of AGV Scheduling**

Fazlollahtabar and Saidi-Mehrabad (2013) reviewed the literature with respect to the methods employed for optimizing AGVs scheduling at the manufacturing, distribution, transshipment, and transportation systems. They classified the existing methods into simulation studies, mathematical approaches, artificial intelligent-based methods, and metaheuristic methods. Generally, the optimization methods are categorized into three approaches of exact, heuristics, and metaheuristic. The exact techniques strive for universal optimality and they generally fail to offer good solutions for NP-hard problems despite the fact that numerous counter examples exist. On the other hand, heuristics are problem-specific methods that exploit the problem properties to draw solution strategies while the metaheuristics are generic heuristic plans that may be applied to numerous optimization problems.

Mallikarjuna (2014) classified the scheduling methods into two categories of traditional and non-traditional methods.

(1) Traditional methods (also referred to as optimization methods)

These methods are generally slow and they only warrant global convergence when the problems under consideration are small. They employ mathematical programming approaches such as integer programming, dynamic programming, linear programming, and transportation programming (e.g., enumerate procedure decomposition like Lagrangian Relaxation).

(2) Non-traditional techniques (also known as approximation methods)

These techniques are very quick but they do not warrant optimum solutions. Some of the approximation methods are as follows:

- a- Constructive methods (e.g., composite dispatching rules and priority dispatch rules),
- b- Insertion algorithms (e.g., shifting-bottleneck procedures and bottleneck-based heuristics),
- c- Evolutionary programs (e.g., particle swarm optimization and genetic algorithms),
- d- Local search techniques (e.g., simulated annealing, ant colony optimization, problem space methods, Tabu search, and adaptive search), and
- e- Iterative methods (These include artificial intelligence methods, artificial neural networks, beam-search, heuristic procedures, and hybrid techniques).

Table 2.1 presents some of the scheduling literature published since 2000, and introduces their objectives and methodologies. It also shows the tendency toward multi-task scheduling of AGV systems and FMSs, in which the makespan minimization criterion is accompanied with several other criteria to entertain an actual-practice scheduling e.g. (Liang et al., 2012; Novas & Henning, 2014; Zhao et al., 2013). Heuristic techniques and EAs are the common optimization methods used to solve a multi-task scheduling problem (Table 2.1). In addition, Table 2.1 introduces the prominent researches in scheduling

context that can assist new researchers in finding a pertinent literature to their specific objective and methodology.

Table 2.1: AGV scheduling literature (objectives and methods)

<b>Authors</b>	<b>The article objectives</b>	<b>Method</b>
Ventura et al. (2015)	Minimization of the response time (mean response time, maximum response time with and without considering time restrictions on vehicle availability).	Mixed integer linear programming formulations + a generic GA
Saidi-Mehrabad et al. (2015)	Minimizing the total completion time, considering the Conflict-Free Routing Problem (CFRP) and the basic Job Shop Scheduling Problem.	A two stage Ant Colony Algorithm (ACA)
Rashmi and Bansal (2014)	Optimal scheduling based on workload balance and minimum traveling time.	Ant colony optimization
Vasava (2014)	Multiple-input job AGV scheduling according to FMS environment.	GA
Wang et al. (2014)	Scheduling for minimization of number of AGVs in the plant.	Simulation
Cai et al. (2014)	Task scheduling and coordination control in a multi-AGV system to shorten the overall run time of the system, and maximize the efficiency of AGVs and overall system.	Mixed regional control model and the neuro-endocrine coordination
Nageswararao et al. (2014)	An autonomous conveyance system for AGVs following the taxi transportation strategies.	Applying traffic engineering knowledge
Nageswararao et al. (2014)	Robust factor function and minimization of mean tardiness	Binary particle swarm Vehicle Heuristic algorithm
Kaplanoğlu et al. (2014)	Proposed a multi-agent based scheduling approach, AGV breakdowns considered.	The Prometheus Methodology
Zeng et al. (2014)	Solved an extension of the blocking job shop (BJS) problem, where transferring jobs between different machines using a limited number of AGVs is concerned (BJS-AGV problem).	A two-stage heuristic algorithm (improving timetabling + local search)
Lin et al. (2014)	Optimal AGV configuration to reduce waiting time.	Simulation
Giglio (2014)	Scheduling the transportation of pallet and roll pallet loads from the storage area to the gates.	Mathematical programming, heuristic procedure
Fazlollahtabar and Shafieian (2014)	Design of a computer integrated manufacturing system Identification of an optimal path in a vehicle routing problem (VRP) network, considering time, cost, and the AGV capability factors.	Mathematical programming approach
Novas and Henning (2014)	Simultaneous scheduling of AGVs, machine loading, manufacturing activities, part routing, machine buffer, and tool planning and allocation in FMS.	Constraint programming
Zhao et al. (2013)	Multi-task scheduling and controlling of the logistic equipment of the AGVS.	Simulation
Sawada et al. (2013)	Scheduling with focus on AGVs congestion at transport rail junctions, throughput maximization, and transit time shortening.	Visualization algorithm via state space realization
Zheng et al. (2013)	Optimizing the AGV running time, minimizing the waiting time, and resolving the conflict and the deadlock problem of the multi-AGV systems.	Mathematical modelling, validated in a test bed.
Ullrich (2013)	Total tardiness minimization through integrating production and outbound distribution scheduling.	GA
Ren et al. (2013)	Study of productivity efficiency in a Collaborative Manufacture System.	Improved GA (coding, crossover, and mutation)
Gan et al. (2013)	AGVs scheduling and comparison with dispatching rules.	Annealing GA

Table 2.1, Continued

Huang and Zhang (2013b)	Optimal AGV scheduling, considering system response time and efficiency.	Game theory
Liang et al. (2012)	Minimization of the make-span considering the AGVs dispatch.	Particle swarm optimization (PSO)
Erol et al. (2012)	Developing an on-line and distributed scheduling system based on a Multi agent-based system (MAS) framework for both AGVs and machines.	Multi-agent based systems (a distributed artificial intelligence technique)
Ariffin et al. (2011)	Minimization of the make-span.	Fuzzy GA
Salehipour et al. (2011)	Locating workstations in a TAGV system using a new solution framework.	Mathematical formulation + development in a heuristic algorithm
Kato and Shin (2010)	Optimal scheduling of the dispatching commands, minimal number of AGVs, and empty load travelling time.	Multi-step solution algorithm
Fauadi and Murata (2010)	Makespan minimization through simultaneous scheduling of machines and AGVs operation.	Binary PSO
Morandin et al. (2010)	Minimization of the makespan by considering AGV and the input buffer.	Timed Petri net
Khanmohammadi et al. (2010)	Proposing a path planning method for AGVs to navigate in an unknown environment reaching certain destinations.	A hybrid Fuzzy logic-based method
Udhayakumar and Kumanan (2010)	Developing a methodology to balance the number of tasks given to the AGVs to minimize the total transportation time (two AGVs were considered)	GA and ant colony
Yahyaei et al. (2010)	Application of a newly developed controller.	Fuzzy logic
Shirazi et al. (2010)	Minimizing the material flow intra- and inter-loops considering the limitation of TAGV workload.	Modified ant colony
Farahani et al. (2008)	Minimizing the maximum workload of the system.	Tabu search and GA
Tavakkoli-Moghaddam et al. (2008)	Minimizing both intra- and inter-loop flows simultaneously based on balanced-loops strategy and inter-machine flows taken from ideas of cellular manufacturing systems (CMS).	Simulated annealing (SA)
Jerald et al. (2006)	Minimization of the penalty cost of machine and its idle time.	Adaptive GA
Jerald et al. (2005)	Minimization of the machine's idle time and total penalty cost for not meeting the deadline concurrently.	PSO
Gaur et al. (2003)	Developing a 5/3 approximation algorithm to minimize the completion time such that each site is visited only after its release time and handling times being taken into consideration.	Dynamic programming strategy, One End first strategy
Sinriech and Kotlarski (2002)	To schedule multiple-load vehicles in a single loop while minimizing the transfer time of jobs and the number of loops travelled by the vehicle.	Dynamic Scheduling (Short term Scheduling algorithm), evaluated through simulation
Berman and Edan (2002)	Developing a control methodology for decentralized autonomous AGVS considering all aspects of AGVS functionality.	Fuzzy control using a hierarchical behavior-based model
Veeravalli et al. (2002)	Proposing analytical models for the AGVs scheduling.	Mathematical modeling

## 2.5 Evolutionary Algorithm

Evolution as proposed by Charles Darwin is a process in nature, where individuals adapt to their environment by preserving the features that makes the individual to compete with others in order to survive and by attempting to eliminate the features that makes it weaker.

Metaheuristic techniques and evolutionary algorithms (EA) are the common optimization methods used to solve a multi-task scheduling problem (Rashmi & Bansal, 2014; Ventura et al., 2015; Wang & Chan, 2014). Some of the distinct researches in scheduling context that can assist new researchers in finding a pertinent literature to their specific objectives and methodologies are (Cai et al., 2014; Kaplanoglu et al., 2014; Nageswararao et al., 2014; Suzuki et al., 2014).

Based on the “no free lunch” (NFL) theory every algorithm can be a suitable choice for a specific class of problems while it may not be a good choice for other classes (Wolpert & Macready, 1997). Literature has shown the effectiveness of genetic algorithms (Pezzella et al., 2008; Zhang et al., 2011) and PSO (Girish & Jawahar, 2009; Zhang et al., 2009) for solving the scheduling problem. Performance of EAs can be improved by the proper choice of their operators and parameters. In addition, hybridization of these algorithms may also further improve their performances. Therefore, GA and PSO algorithms are used in this study. However, the hybrid of GA and PSO is commonly believed to be more effective than its constituent algorithms (Mehta, 2012; Wang & Si, 2010; Wu et al., 2010). Although literature has practiced hybridization of the GA and PSO, due to the many possibilities in the choice of operators and parameters integration strategy in the hybrid form, it is always novel to find a better strategy for obtaining the optimum result in a specific problem.

### **2.5.1 Genetic Algorithm**

GA is a search algorithm based on the mechanics of the natural selection process (biological evolution) which were pioneered by John Holland and his students (Holland, 1975). GA is popular among all the evolutionary algorithms and they are applied successfully to a variety of engineering problems of different fields. The most basic concept in GA is that the strong tends to adapt and survive while the weak tends to vanish.

Genetic algorithms are stochastic heuristic search methods whose mechanisms are based upon simplifications of evolutionary processes observed in nature. They are good at both the exploration and exploitation of the search space, as they operate on more than one solution at any instance of time (Wall, 1996).

Genetic algorithm works by modeling the parameters of a problem as bit strings. They may represent integers, floating-point numbers, or anything else that is applicable to a problem. Each of these parameters is referred to as a gene and the bit string as chromosome in the context of GA. Initially a population of chromosomes, each of which represents a potential solution to the problem at hand, is generated randomly and each of them is evaluated based on its fitness value. The next generation of same size is created by selecting more fit individuals from this population and by applying genetic operators like crossover and mutation to them. Mutation is an operator, which creates a new individual by making a random change in the old one, whereas crossover creates new individuals by combining parts from multiple individuals. Classic mutation randomly alters a single gene, while crossover exchanges genetic material between two or more parents. This completes one generation and after repeating this procedure for a number of generations, due to the selection operator utilization, the algorithm converges and it yields a better solution (Aytug et al., 2003).

GA has been successfully applied in many of the scheduling studies. Biegel and Davern (1990) applied GA to the scheduling problem and discussed the GA process for an elementary “n” tasks one-machine problem. Then they extended the work for “n” tasks on two processors and finally generalized for “n” tasks and “m” processors in the series. Chen et al. (1995) proposed a GA based heuristic for the scheduling problem on the makespan objective and compared the efficiency of the proposed GA with the other GA heuristics reported in the literature. A thorough review of the GA representation schemes

for scheduling and various hybrid approaches of GA and conventional heuristics application is prepared by Cheng et al. (1996).

Some of the studies in this context has been working on improving the GA application in the scheduling problems. For instance, Nearchou (2004) investigated the effect of various genetic operators on the performance of GAs when applied on permutation scheduling problems. The stochastic behavior of the GA was estimated under the influence of a set of five crossover and six mutation schemes. To study the GA hybridization possibilities, Gonçalves et al. (2005) have developed a hybrid genetic algorithm for the scheduling problem that the chromosome representation of the problem was based on random keys. Their scheduling model was constructed using a priority rule in which the priorities were defined using GA.

Udhayakumar and Kumanan (2010) studied multi-objective task scheduling of AGVs in an FMS using non-traditional optimization algorithms. They tried to find a near optimum schedule for two AGVs based on the balanced workload and minimum traveling time for maximum utilization. Agrawal et al. (2012) applied GA for a multi-objective scheduling problem where alternate machines were available to process the same job to minimize makespan as well as total machining time. The application of GA to the multi-objective scheduling problem has given optimum solutions for allocation of jobs to the machines to achieve nearly equal utilization of machine resources. Vasava (2014) used GA to develop an AGV scheduling model for different FMS environments with the objective of makespan minimization.

### **2.5.2 Particle Swarm Optimization**

Particle swarm optimization (PSO) is an evolutionary algorithm that was inspired by the motion of a flock of birds searching for foods and was proposed by Kennedy and Eberhart (1995). PSO is one of the successful optimization algorithms in scheduling applications

because of its simple and straightforward implementation. PSO has been applied in multiple fields such as human tremor analysis for biomedical engineering, electric power and voltage management and machine scheduling (Kennedy et al., 2001). The original PSO is proposed for optimization of single objective continuous problems. However the concept of PSO has been expanded to allow it to handle other optimization problems such as; binary, discrete, combinatorial, constrained and multi-objective optimization (Aziz et al., 2011).

At the beginning of the evolutionary process, a set of particles referred to as a swarm must be initiated randomly. Each particle can change its position in the search space just like a flying bird searching the food in the sky. During the evolutionary process, a particle of a swarm adjusts its newer moving velocity according to its best experience, the best experience of all particles in the swarm and the previous moving velocity. Then, the particle moves to a new position according to newly generated velocity and its previous position (Zini & ElBernoussi, 2015).

Some of the studies that have applied PSO for scheduling problems are reviewed in the followings. Tasgetiren et al. (2006) applied PSO to solve the single machine total weighted tardiness problem, which is a typical discrete combinatorial optimization problem. They developed a heuristic rule borrowed from the random key representation in genetic algorithms, called ‘the smallest position value’ (SPV) rule to enable the continuous PSO to be applied to all permutation types of discrete combinatorial optimization problems. Tasgetiren et al. (2007) applied PSO algorithm to solve the permutation-scheduling problem with the objectives of minimizing makespan and the total flow time of jobs. Performance of the algorithm was evaluated on widely used benchmarks from the operations research library and it showed to be promising in solving permutation problems. In the same vein, Chandrasekaran et al. (2007) developed a PSO



to solve a multi-objective scheduling problem. They had considered three criteria of minimization of makespan, total flow-time, and the completion time variance. Performance of the proposed methodology has been tested by solving benchmark scheduling problems available in the literature.

Pongchairerks (2009) introduced three heuristic algorithms based on PSO for solving scheduling problems with multi-purpose machines, and open-shop scheduling problems. The three developed algorithms were based on the PSO and specific decoding procedures generating solutions related to the class of parameterized active schedules. Pongchairerks (2009) concluded the superiority of developed algorithms against available ones. Later, Sha and Lin (2010) constructed a PSO for an elaborate multi-objective job-shop scheduling problem. Due to the discrete solution spaces of scheduling optimization problems, they modified the particle position representation and particle movement and velocity. They used the modified PSO to solve various benchmark problems. They proved that their modified PSO performed better in search quality and efficiency than traditional evolutionary heuristics.

Recently, Al Theeb and Alhwiti (2014) used PSO to minimize the total weighted tardiness and the total setup costs for all jobs, where the setup time and cost for a job depend on its place in the sequence. They highlighted that there is no relationship between the setup time and cost and this is because in some cases the setup requires highly skilled operators and special tools, which cause the setup cost to be high even if the setup time is short.

### **2.5.3 Hybrid Algorithms**

Hybrid algorithms are constructed by combining two or more other algorithms that originally solve the same problem. So that for solving the problem throughout the hybrid algorithm run, either one (based on the data properties) algorithm will be used or the hybrid algorithm switches between its constituent algorithms. The hybridization approach

is to benefit from desired features of each algorithm in a compound that can have a better performance than its constituent algorithms. It should be stressed that a hybrid algorithm is an algorithm that is built by combining several other algorithms that solve the same problem—not just any algorithms that may solve other problems—but each has different characteristics like performance.

- Hybrid GA and PSO

The PSO algorithm is problem-independent, which means little specific knowledge relevant to a given problem is required (Xia & Wu, 2006). All the required prior-knowledge is the fitness evaluation of each particle. This advantage makes PSO more robust than many other search algorithms. On the other hand, GA simultaneously evaluates many points in the search space, it is more likely to find the global solution of a given problem. With every EA having a specific merit that could be incorporated into another EA, hybridization of evolutionary algorithms has been investigated in many studies (Chelouah & Siarry, 2003; Fan & Zahara, 2007; Kao & Zahara, 2008). Such a hybrid is often referred as a mimetic algorithm.

As known, PSO performs according to the knowledge of social interaction, and all individuals are taken into account in each generation. On the contrary, GA simulates evolution and some fitter chromosomes are selected while some others are eliminated from generation to generation (Liou et al., 2013). Hence, by integrating the advantages of the compensatory properties of PSO and GA, their hybrid form can be used to obtain better results (Mehta, 2012; Wang & Si, 2010; Wu et al., 2010).

For example, Tang et al. (2010) proposed a hybrid of GA and PSO to solve the scheduling problem. The PSO algorithm was introduced to get the initial population, while evolutionary genetic operations were used. They validated the new method on seven benchmark datasets, and the comparison with some existing methods verified its

effectiveness. Liu et al. (2015) used a hybrid algorithm using PSO and GA to solve a scheduling problem. In their algorithm, named hybrid PSO-GA algorithm (HPGA), the PSO algorithm was redefined and modified by introducing genetic operators, i.e. the crossover operator and the mutation operator, to update the particles in the population. The HPGA was then applied in heavy machinery company with minimizing machines' makespan and minimizing jobs' tardiness as the two optimal objectives. The comparisons with actual application report showed that their proposed HPGA can obtain higher quality of schedule solution for machine tool production.

#### **2.5.4 Multi-objective optimization**

One of the most widely used methods for solving multi-objective optimization problems is the classical method. Such methods aggregate the objectives into a single, parameterized objective function in order to generate the Pareto-optimal set. Several optimizations run with different parameter settings are performed in order to achieve a set of solutions which approximates the Pareto-optimal set. Basically, this procedure is independent of the underlying optimization algorithm. Some representatives of this class of techniques are the weighting method (Cohon 1978), the constraint method (Cohon 1978), and etc.

The fact that well-studied algorithms for single-objective problems can be used for multi-objective problems makes the classical approaches attractive and popular. For large-scale problems, hardly any real multi-objective optimization techniques had previously been available (Horn 1997). By contrast, in single-objective optimization a wide range of heuristic methods have been known that are capable of dealing with this complexity (Zitzler, 2000, 1999, Mao-Guo, 2009. Deb, 2015).

The weighted-sum method is a traditional and a popular method that parametrically changes the weights among objective functions to obtain the Pareto front (Kim, 2005). It

is simple and easy to use, for convex problems it guarantees finding solutions on the entire Pareto-optimal set, although it has a limitation in mixed optimization problems (min-max) and needs to have all the objectives in one type. However, as objectives of this study are of minimization type, the weighted sum method can properly be applied in this research framework.

## 2.6 The current research establishment

Based on the reviewed literature relevant to the present research, Table 2.2 is prepared to demonstrate the methods, model criteria, and validation procedure used in this study and compare it with existing literature for establishing its novelty.

Table 2.2: Studies with similar research components

	Method			Model					Method			Model			
	1	2	3	4	5	6	7		1	2	3	4	5	6	7
Ventura et al. (2015)	✓							Liang et al. (2012)		✓		✓	✓		
Saidi et al. (2015)				✓				Morandin et al. (2011)	✓						✓
Samuel and Rajan (2015)			✓					Fauadi and Murata (2010)		✓		✓			
Jain and Foley (2016)							✓	Kato and Shin (2010)					✓		
Liu et al. (2015)			✓	✓				Udhayakumar and Kumanan (2010)	✓						
Li et al. (2014)				✓				Tang et al. (2010)			✓				
Vasava (2014)	✓							Yahyaei et al. (2010)		✓					
Wang et al. (2014)					✓		✓	Song (2010)			✓				
Lin et al. (2014)							✓	Kuo and Lin (2010)			✓				
Giglio (2014)				✓				Premalatha and Natarajan (2010)			✓				
Nageswararao et al. (2014)		✓						Gnanavel Babu et al. (2010)				✓			
Novas and Henning (2014)				✓				Sha and Lin (2010)		✓					
Zheng et al. (2014)				✓				Morandin et al. (2010)				✓			
Ercan and Li (2013)			✓					Naderi et al. (2010)				✓			
Jamrus et al. (2013)			✓					Premalatha and Natarajan (2009)			✓				
Zhao et al. (2013)							✓	Valdez et al. (2008)			✓				
Zheng et al. (2013)							✓	Farahani et al. (2008)	✓						
Ullrich (2013)	✓							Kim et al. (2007)			✓				
Gan et al. (2013)	✓							Jerald et al. (2006)	✓						
Huang and Zhang (2013a)							✓	Reddy and Rao (2006)				✓			
Gelareh et al. (2013)				✓				Jerald et al. (2005)		✓					
Badakhshian et al. (2012)				✓				Abdelmaguid et al. (2004)				✓			
Udhayakumar and Kumanan (2012)				✓				Gaur et al. (2003)				✓			
Liang et al. (2012)		✓	✓					Haq et al. (2003)				✓			
Agrawal et al. (2012)	✓							Sinriech and Kotlarski (2002)							✓
Badakhshian et al. (2012)	✓			✓				Veeravalli et al. (2002)							✓
1- GA 2- PSO 3- Hybrid GA and PSO 4- AGV Scheduling (makespan minimization)								5- AGV Scheduling (AGV number minimization) 6- AGV Scheduling (AGV battery charge consideration) 7- Simulation of model							

Table 2.2 demonstrates the study components of previous researches in scheduling context, where it is the focus of this dissertation as well. The research plan of this study was to define a set of criteria for scheduling model development that can significantly contribute to the scheduling practices. Previous studies, as summarized in table 2.2, had not considered AGV battery charge in their scheduling models (column number 6 in gray), in which it can affect the scheduling models' practicality. In addition, no study had also practiced a combination of these seven components stated in Table 2.2 for AGV scheduling. Therefore, this study by integrating the above study components can provide a significant contribution and a novel approach in the AGV scheduling context.

## **2.7 Summary**

A flexible manufacturing system is a “reprogrammable” automated manufacturing system capable of producing a variety of products. In an FMS, various machining cells are interconnected via loading/unloading stations by an automated transportation system such as AGV system. AGV systems are one of the skillful types of material handling system in modern automated production environments. AGVs are the ideal equipment for warehouse storage, automatic loading/unloading of trailers with pallets and other unit loads, and production applications. Some of the main advantages of AGVs are the high flexibility, space utilization, and safety along with less overall operating cost. For cost-effective utilization of AGVs, proper planning on AGVs dispatching, scheduling and routing should be considered (Le-Anh & De Koster, 2006). The term ‘scheduling’ refers to the process of allocating AGVs to tasks, taking into account the costs and time of operations (Udhayakumar & Kumanan, 2010). Selection of the right scheduling policy highly influences the FMS performance. AGVs—are expensive devices and highly valuable resource in FMSs that their operation should be adequately optimized for the system to be profitable (Anwar & Nagi, 1998; Fauadi & Murata, 2010; Nanvala, 2011). Literature review revealed that the potential of AGV scheduling with objective setting of

minimizing the number of AGVs and makespan while considering the AGVs' battery charge has not been studied yet. Thus, considering the importance of each of the above criteria in FMS profitability, this study developed a scheduling model with the above three criteria. In addition, evolutionary algorithms have been proved a powerful tool for scheduling optimization problems. Overall performance of an evolutionary algorithm, like PSO and GA, can be improved by proper choice of its operators and parameters (Gen & Lin, 2014). Hybridization of evolutionary algorithms is also believed that can profoundly enhance the overall performance of the EAs. PSO functions according to the knowledge of social interaction, and all the individuals are taken into account in each generation. On the contrary, GA simulates the evolution and some fitter chromosomes are selected while some others are eliminated from generation to generation. Integrating advantages of the two EAs of GA and PSO in a single optimization algorithm, results in a hybrid algorithm that can be more beneficial than its constituents. There are many possibilities in the choice of operators and parameters integration strategy in the hybrid form that makes it novel to find a better strategy for obtaining the optimum result in a specific problem. Thus, hybridization of two well-known algorithms of GA and PSO through a new integration approach (called HGP2) is accomplished for model optimization in this study.

## **CHAPTER 3: METHODOLOGY**

### **3.1 Introduction**

This chapter describes the research framework of the study as graphically summarized and shown in Figure 3.1. Having accomplished a thorough literature review, the scheduling model derivation, its assumptions and objective criteria are explained first. The general and detailed flowchart of the model are illustrated in Figures 3.2 and 3.3, respectively. Next, the four evolutionary algorithms (GA, PSO, and 2 different hybrids of GA-PSO referred to as HGP1 and HGP2) developed for the model are elaborated through. A detailed description on AGVs specification/behaviour exploration is then provided to be applied both before and after optimization. Model validation using testbed run and simulation technique through FlexSim software are explained at the end of this chapter.

### **3.2 Research Framework**

The research framework is shown in Figure 3.1. In this figure, the blue rectangular represents the first research objective which is multi-objective model development, and its details are demonstrated in Figures 3.2 and 3.3. The red rectangular shows the processing blocks of the second objective of algorithms development, which further details are illustrated in Figures 3.5, 3.11, 3.12, and 3.13. The third objective is the last part of the framework which is testbed run and simulation.

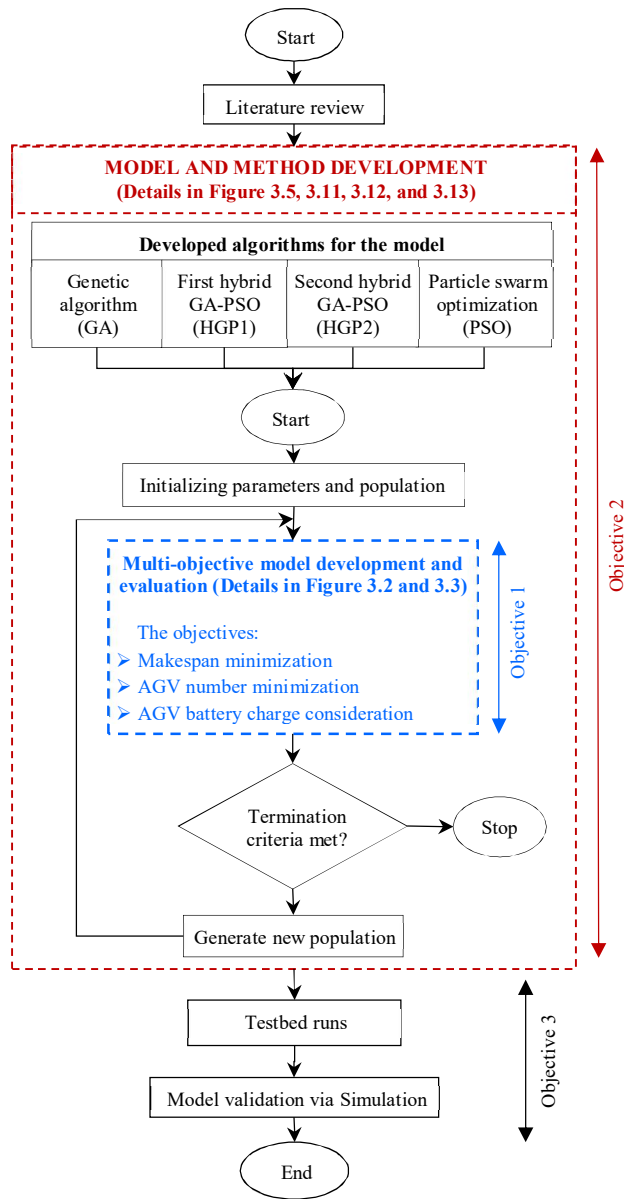


Figure 3.1: The overall research framework

### 3.3 Model Derivation

This section explains the mathematical model development for AGV scheduling using the three criteria selected based on the reviewed literature. The three criteria are categorized into two main objectives: (1) minimizing the makespan and (2) minimizing the number of AGVs while considering the AGV's battery charge. First, it is necessary to define the conditions and limitations considered in the model development. Thus, the following conditions were defined:



- All AGVs have unit-load capacity and it is same in whole procedure.
- AGVs and machines operate continuously without breakdown.
- Traffic problems, collision, or conflicts are avoided by hardware and are not considered in this study.
- AGV L/U times are fixed and considered in travel times.
- AGVs are allowed to park at their P/D locations.
- AGVs have a constant speed and move forward only
- L/U equipment such as pallets are sufficiently allocated as well as output buffer for machines to avoid machine deadlock.
- The machine-to-machine distance and L/U point-to-machine distances are known.
- Each machine operates only one product at a time.

Having defined the conditions for the model development, Figure 3.2 illustrates the six general processing blocks/steps of the model development. It starts with inspection of AGVs' status and their current position in the layout; the operations' order would also be reviewed simultaneously. AGV selection based on battery status and travel time of current AGVs in the system is then performed and it is compared with the possibility of adding a new AGV to the system. The selected AGV is next assigned to the task. Final, machines and parts availability would be reviewed for the AGV to perform the loading/unloading of the part.

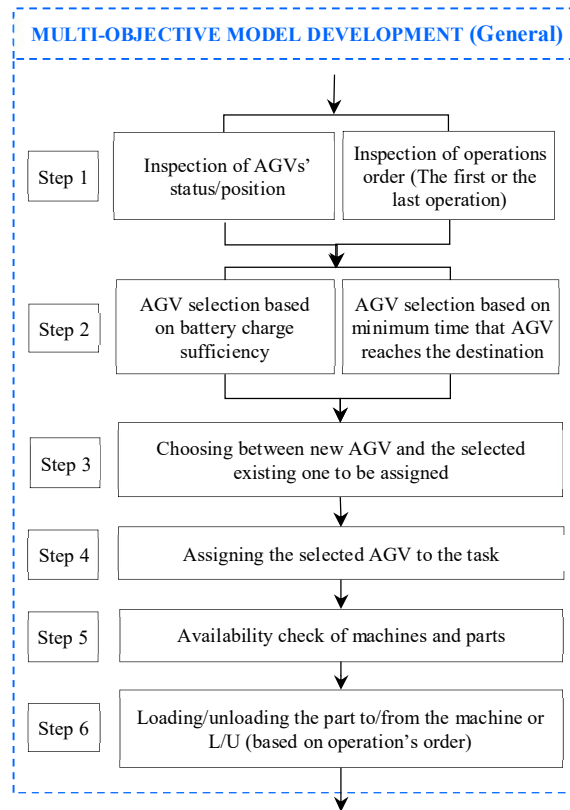


Figure 3.2: General flowchart of the multi-objective optimization model

For further explanations on each step, Figure 3.3 illustrates the detailed flowchart of model development. The pseudocode of the model development algorithm is also shown in Figure 3.4 for a straightforward pursuant of the flowchart.

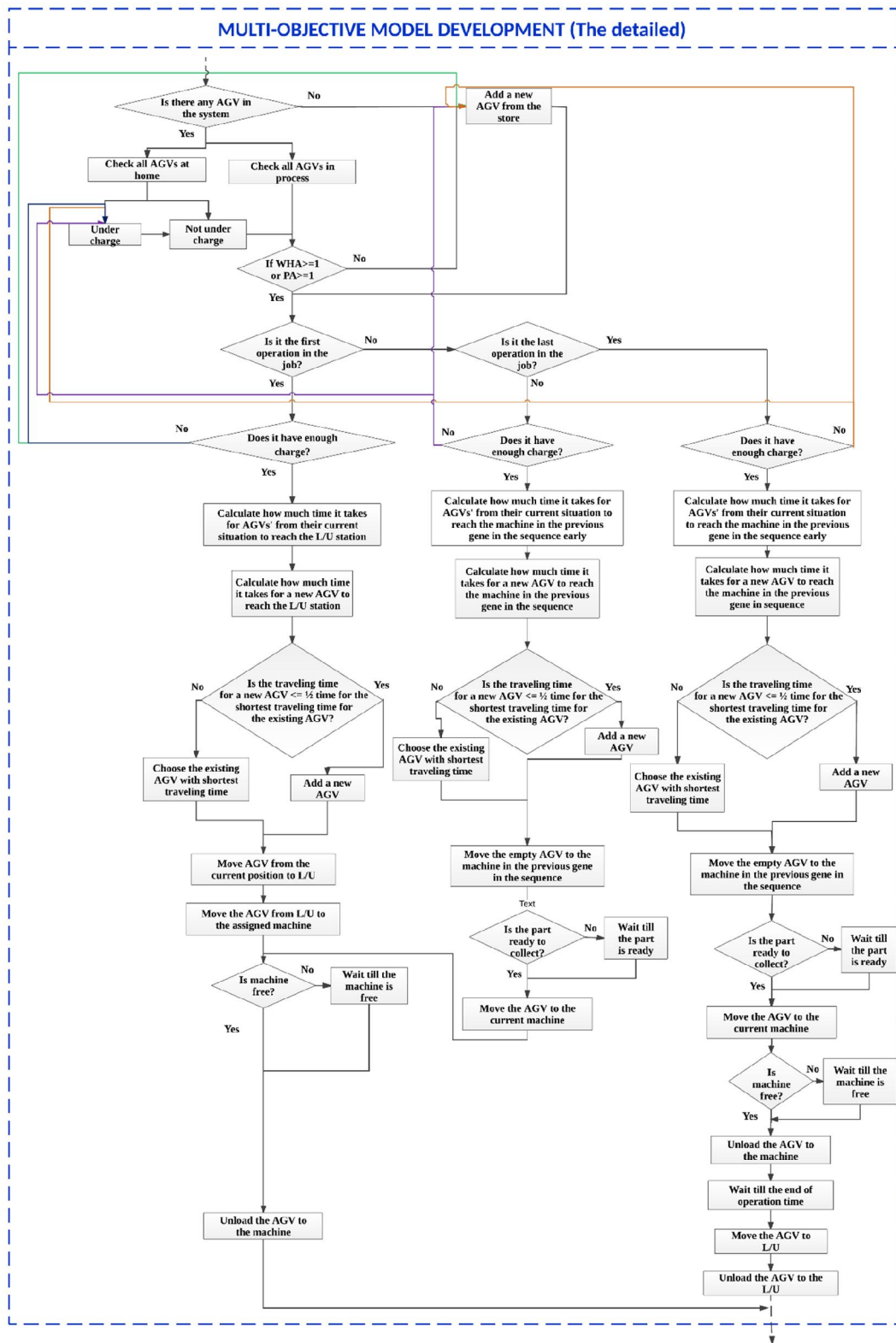


Figure 3.3: The detailed flowchart of the multi-objective optimization model

```

Start
if (active AGV in the system) then
  for (all AGVs in the system (at home (under charge or fully charged)/all AGVs in the process))
    if (AGVs >=1) then
      for (all operations)
        if (operation is the first operation of the job) then
          for (all AGVs check the charge)
            if (has enough charge) then
              for (all AGVs with enough charge)
                Calculate how much time it takes for AGVs' from their current situation to
                reach the L/U station
                Calculate time for new AGV to reach L/U
                if (the traveling time for a new AGV <= ½ time for the shortest traveling time
                for the existing AGV) then
                  Add a new AGV
                  Move AGV from the current position to L/U
                  Move the AGV from L/U to the assigned machine
                  if (machine is free) then
                    Unload the AGV to the machine
                  else
                    Wait till machine is free then
                    Unload the AGV to the machine
                  end if
                else
                  Choose the existing AGV with shortest traveling time
                  Move AGV from the current position to L/U
                  Move the AGV from L/U to the assigned machine
                  if (machine is free) then
                    Unload the AGV to the machine
                  else
                    Wait till machine is free then
                    Unload the AGV to the machine
                  end if
                end if
              end for
            end for
          else
            for (all AGVs with not enough charge)
              Send AGV home to charge
              Add a new AGV from the store
            end for
          end if
        end if
      end for
    end if
  end if

```

Figure 3.4: Pseudocode of the model

```

end for
else if (it is the last job) then
  for (all AGV's check the charge)
    if (has enough charge) then
      for (all AGV's with enough charge)
        Calculate how much time it takes for AGV's' from their current situation
        to reach the machine in the previous gene in the sequence early
        Calculate how much time it takes for a new AGV to reach the machine
        in the previous gene in the sequence
        if (the traveling time for a new AGV  $\leq$  ½ time for the shortest
        traveling time for the existing AGV) then
          Add a new AGV
          Move the empty AGV to the machine in the previous gene in the
          sequence
          if (part ready to collect)
            Move the AGV to the current machine
            if (machine is free) then
              Unload the AGV to the machine
            else
              Wait till machine is free then
              Unload the AGV to the machine
              Wait till the end of operation time
              Move the AGV to L/U
              Unload the AGV to the L/U
            end if
          else
            Wait till part is ready
            Move the AGV to the current machine
            if (machine is free) then
              Unload the AGV to the machine
            else
              Wait till machine is free then
              Unload the AGV to the machine
              Wait till the end of operation time
              Move the AGV to L/U
              Unload the AGV to the L/U
            end if
          end if
        else
          Choose the existing AGV with shortest traveling time
          Move the empty AGV to the machine in the
          previous gene in the sequence
        end if
      end if
    end for
  end if

```

Figure 3.4: Continued

```

    if (part ready to collect)
        Move the AGV to the current machine
        if (machine is free) then
            Unload the AGV to the machine
        else
            Wait till machine is free then
            Unload the AGV to the machine
            Wait till the end of operation time
            Move the AGV to L/U
            Unload the AGV to the L/U
        end if
    else
        Wait till part is ready
        Move the AGV to the current machine
        if (machine is free) then
            Unload the AGV to the machine
        else
            Wait till machine is free then
            Unload the AGV to the machine
            Wait till the end of operation time
            Move the AGV to L/U
            Unload the AGV to the L/U
        end if
    end if
end if
end for
else
    Send current AGV to the L/U to charge
    Add a new AGV from the store
end if
end for
else
    for (all AGVs check the charge)
        if (has enough charge) then
            for (all AGVs with enough charge)
                Calculate how much time it takes for AGVs' from their current situation to
                reach the machine in the previous gene in the sequence early
                Calculate how much time it takes for a new AGV to reach the machine in
                the previous gene in the sequence
                if (the traveling time for a new AGV  $\leq$  ½ time for the shortest traveling time
                for the existing AGV) then
                    Add a new AGV
                end if
            end for
        end if
    end for
end else

```

Figure 3.4: Continued

```

Move the empty AGV to the machine in the previous gene in
the sequence
if (part ready to collect)
    Move the AGV to the current machine
    if (machine is free) then
        Unload the AGV to the machine
    else
        Wait till machine is free then
            Unload the AGV to the machine
    end if
    end if
end if
end for
end if
end for
end if
    Add a new AGV from the store
end for
else
    Add a new AGV from the store
end if

```

Figure 3.4: Continued

In following sections, mathematical definitions used to develop the model are explained. In the first section, since the model has more than one objective, the objectives combining approach for the purpose of model evaluation is described. The later subsections would discuss the objectives individually.

### 3.3.1 Multi-objective Evaluation

There are different ways of evaluating multi-objective models. Weighted sum is one of the most applied principles for evaluation of multi-objective optimization problems (Karthikeyan et al., 2015; Marler & Arora, 2010). In this method, different objective values are aggregated into a single quality measure. Objective functions usually have different scales from one another and in weighted sum approach, objectives would be normalized prior to the aggregation process (Eichfelder, 2008; Giagkiozis & Fleming,

2015). Based on the weighted sum method, overall fitness function formulation for  $\beta$  objectives is described by

$$f(x) = \delta_1 f_1(x) + \dots + \psi_\beta \delta_\beta f_\beta(x) \quad \wedge \quad \delta_1 + \dots + \delta_\beta = 1 \quad \wedge \quad \delta_\beta > 0 \quad (3.1)$$

where  $\beta$  is the index of  $\delta$ , and  $\beta = 1, \dots, L$ ,  $\delta_\beta$  is the  $\beta^{th}$  weight of the  $\beta^{th}$  objective function, and  $\psi$  is a ratio to make balance among objectives with different ranges of value (Ghane-Kanafani & Khorram, 2015; Mateo, 2012), which is defined by

$$\psi_\beta = \frac{\max f_1(x)}{\max f_\beta(x)} \quad \text{when } f_1(x) \gg f_\beta(x) \quad (3.2)$$

### 3.3.1.1 Minimizing the Makespan

This step involves calculating makespan ( $MS$ ) which is the time required for all operations to be completed. A set of  $n$  jobs denoted by  $J_{j'}$  has some operations denoted by  $O_{ji}$  (operation  $i$  of job  $j$ ), which will be processed on a set of machines ( $M_{ji}$ ). A general schematic for reading data is shown in Table 3.1. Makespan is expressed by

$$MS = \max \{ (tDT_{ji}^a + p_{ji}) \} \quad (3.3)$$

$$tDT_{ji}^a = tCA^a + UT_{ji}^a + WT_{ji}^a + LT_{ji}^a \quad (3.4)$$

$$WT_{ji}^a = \begin{cases} p_{j(i-1)}^e - rPT_{ji}^a & \text{if } rPT_{ji}^a < p_{j(i-1)}^e \wedge rDT_{ji}^a \geq p_{j'i'}^e \wedge i \neq 1 \\ (p_{j(i-1)}^e - rPT_{ji}^a) + (p_{j'i'}^e - rDT_{ji}^a) & \text{if } rPT_{ji}^a < p_{j(i-1)}^e \wedge rDT_{ji}^a < p_{j'i'}^e \wedge i \neq 1 \\ p_{j'i'}^e - rDT_{ji}^a & \text{if } \begin{cases} rDT_{ji}^a < p_{j'i'}^e \wedge i = 1 \\ \vee \\ rDT_{ji}^a < p_{j'i'}^e \wedge rPT_{ji}^a \geq p_{j(i-1)}^e \wedge i \neq 1 \end{cases} \\ 0 & \text{if } \begin{cases} rDT_{ji}^a \geq p_{j'i'}^e \wedge i = 1 \\ \vee \\ rDT_{ji}^a \geq p_{j'i'}^e \wedge rPT_{ji}^a \geq p_{j(i-1)}^e \wedge i \neq 1 \end{cases} \end{cases} \quad (3.5)$$

Subject to  $M_{ji} \triangleq M_{j'i'}$



$$UT_{ji}^a = tPT_{ji}^a - tCA^a = \begin{cases} tHT_{ji}^a - tM_{j(i-1)} & \text{if } CA^a = M_{j(i-1)} \wedge i = 1 \\ tHT_{ji}^a - tM_{ji} & \text{if } CA^a = M_{ji} \wedge i = 1 \\ tHT_{ji}^a - tM_{j'i'} & \text{if } CA^a = M_{j'i'} \wedge i = 1 \\ 0 & \text{if } \begin{cases} CA^a = H \wedge i = 1 \\ \vee \\ CA^a = M_{j(i-1)} \wedge i \neq 1 \end{cases} \\ tM_{j(i-1)}T_{ji}^a - tH & \text{if } CA^a = H \wedge i \neq 1 \\ tM_{j(i-1)}T_{ji}^a - tM_{ji} & \text{if } CA^a = M_{ji} \wedge i \neq 1 \\ tM_{j(i-1)}T_{ji}^a - tM_{j'i'} & \text{if } CA^a = M_{j'i'} \wedge i \neq 1 \end{cases} \quad (3.6)$$

$$LT_{ji}^a = tDT_{ji}^a - tPT_{ji}^a = \begin{cases} tM_{ji}T_{ji}^a - tHT_{ji}^a & \text{if } PT_{ji}^a = H \wedge i = 1 \\ tM_{ji}T_{ji}^a - tM_{j(i-1)}T_{ji}^a & \text{if } PT_{ji}^a = M_{j(i-1)} \wedge i \neq 1 \end{cases} \quad (3.7)$$

Subject to:

$$CTO_{ji} \geq p_{ji}^s \quad \forall i = 1 \quad (3.8)$$

$$tPT_{ji}^a \geq 0 \quad \forall T_{ji}^a \in T^a \quad (3.9)$$

$$p_{ji}^s \geq tPT_{j(i+1)}^a - p_{ji} \quad \forall j, i \quad (3.10)$$

$$p_{ji}^s - p_{j(i-1)}^s \geq p_{ji} + LT_{ji}^a \quad \forall j, i = 2, \dots, m_j \quad (3.11)$$

$$\begin{aligned} & (p_{ji}^s - p_{j'i'}^s - p_{j'i'} + \mu |M_{ji} - M_{j'i'}| \geq 0) \vee \\ & (p_{j'i'}^s - p_{ji}^s - p_{ji} + \mu |M_{ji} - M_{j'i'}| \geq 0) \quad \forall (j, i, j', i') \end{aligned} \quad (3.12)$$

$$\begin{aligned} & (tPT_{jm_j}^a - tPT_{j'i'}^a - LT_{j'i'}^a + \mu |T_{ji}^a - T_{j'i'}^a| \geq 0) \vee \\ & (tPT_{j'i'}^a - tPT_{jm_j}^a - LT_{ji}^a + \mu |T_{ji}^a - T_{j'i'}^a| \geq 0) \quad \forall (j, m_j, j', i') \end{aligned} \quad (3.13)$$

where  $m_{j,j'}$  is the total number of operations of job  $j, j'$ ,  $i, i'$  is the indexes of operations,

$(i, i' = 1, 2, \dots, m_{j,j'})$ ,  $n$  is the total number of jobs,  $LT_{ji}^a$  is the loaded time of  $A^a$  doing  $T_{ji}$ .

$p_{ji}$  is the processing time of  $O_{ji}$ ,  $p_{ji}^s$  is the start time of processing time of  $O_{ji}$ ,  $p_{ji}^e$  is the end

time of processing time of  $O_{ji}$ , and  $T_{ji}^a$  is the assigned  $A^a$  to do the task  $T_{ji}$ .  $CTO_{ji}$  is the

completion time of operation  $O_{ji}, \mu$  is a large positive number.  $a, a'$  are indexes of AGVs, ( $a, a' = 1, \dots, z$ ), and  $A^{a, a'}$  represents AGVs.  $PT_{ji}^a$  is the pick-up point of  $A^a$  doing  $T_{ji}$ ,  $tPT_{ji}^a$  is the pick-up time of  $A^a$  doing  $T_{ji}$ ,  $rPT_{ji}^a$  is the time that  $A^a$  reaches pick-up place of  $T_{ji}$ .  $DT_{ji}^a$  is the drop-off point of  $A^a$  doing  $T_{ji}$ ,  $tDT_{ji}^a$  is the drop-off time of  $A^a$  doing  $T_{ji}$ ,  $rDT_{ji}^a$  is the time that  $A^a$  reaches drop-off place of  $T_{ji}$ ,  $UT_{ji}^a$  is the unloaded time of  $A^a$  doing  $T_{ji}$ , and  $WT_{ji}^a$  is the waiting time of  $A^a$  doing  $T_{ji}$ .

Constraint number 3.8 ensures the feasibility of completion time of the first operation of each job. Constraints number 3.9 and 3.10 ensure the feasibility of pick-up time of operations. Inequality number 3.11 describes the operations precedence constraint. Inequalities number 3.12 and 3.13 represent the operation and the AGV un-overlapping constraints, respectively.

### 3.3.1.2 Minimizing the Number of AGVs

This step involves calculating the number of AGVs, which is denoted by  $NA$ . Number of AGV is expressed by

$$NA = \max \{a\} \mid T^a \in T \quad (3.14)$$

Subject to

$$A^a \text{ is assigned to } T_{ji} \text{ (to create } T_{ji}^a) \quad \text{if} \left\{ \begin{array}{l} ChA^a \geq ChHT_{ji}^a \\ \wedge \\ tCPT_{ji}^a < tCPT_{ji}^{a'} \\ \vee \\ tCPT_{ji}^a = tCPT_{ji}^{a'} \quad \wedge \quad a < a' \end{array} \right. \quad (3.15)$$

$$A^y \Big|_{y \text{ is a new AGV}} \text{ is assigned to } T_{ji} \text{ (to create } T_{ji}^y) \quad (3.16)$$

$$\text{if } \left\{ (tCPT_{ji}^y + LT_{ji}^y + UT_{ji}^y) \leq \lambda (tCPT_{ji}^a + LT_{ji}^a + UT_{ji}^a) \right\}$$

where  $T_{ji}$  is the related task to  $O_{ji}$  (moving from  $M_{ji-1}/H$  to  $M_{ji}$ ),  $T^a$  is a collection of operations that have done by  $A^a$ ,  $CA^a$  is the current position of  $A^a$ ,  $tCA^a$  is the time of current position of  $A^a$ , and  $H$  is the loading/unloading (Home) point.  $ChA^a$  is the current battery charge of  $A^a$ ,  $ChHT_{ji}^a$  is the charge that  $A^a$  needed for doing the task  $T_{ji}$  and return home.  $tT_{ji}^a H$  is the time that  $A^a$  arrives home after doing  $T_{ji}$ ,  $y$  is the index of new AGVs.  $\lambda$  is a coefficient for determining when a new AGV should be added, and  $tCPT_{ji}^a$  is the travel time of  $A^a$  from its current point to reach the start point of  $T_{ji}$ .

Equation 3.15 ensures that the assigned AGV has enough battery charge to do the job and return home, while it chooses the AGV which takes less time to reach the point. Equation 3.16 determines the suitable time for adding a new AGV to the system.

#### (a) Battery Charge of AGV

In the model, AGV battery charge has been considered at all time. When an AGV is to be assigned to a job, first its battery charge sufficiency ( $ChHT_{ji}^a$ ) will be checked. AGV should have enough battery to travel and do the job, then return home.

$$ChHT_{ji}^a = \gamma(UT_{ji}^a + LT_{ji}^a + (tT_{ji}^a H - tDT_{ji}^a)) \quad (3.17)$$

$$ChA^a = \sum_{\substack{j,i \\ T_{ji}^a \in T^a}} ChT_{ji}^a \quad (3.18)$$

$$ChT_{ji}^a = \gamma(LT_{ji}^a + UT_{ji}^a) \quad (3.19)$$

$$BU^a = \frac{ChA^a}{\gamma(MS - \overline{TuA^a})} \times 100 \quad (3.20)$$

where  $ChA^a$  is consumed battery charge of AGV number  $a$ ,  $\gamma$  is the ratio of energy consumption to time,  $\overline{TuA^a}$  is the time that AGV is being charged, and  $BU^a$  shows the consumed battery charge utility of AGV number  $a$ . Equation 3.17 calculates the battery charge required for the AGV to do the job and return home. As battery-run-time of an AGV and battery-charging-time can be defined depending on the type of batteries used, charging methods, charge rate, application type, manufacturer, and assignments the vehicles perform,  $\gamma$  has been defined to adopt to any kind of battery and charging method, etc. The “automatic and opportunity” battery-charging is considered here, which on average an AGV charges for 10-12 minutes every hour (AGV Kennis Instituut, 2015; Egemin Automation, 2016).

**(b) AGVs’ Specifications/Behavior**

In this step, the methodology for further investigation on AGVs’ behaviour before and after the optimization is explained. Such analysis would help in better understanding of the optimization model impacts and effectiveness. Specifications of AGV number  $a$  are its total running time denoted by  $RtA^a$  (loaded ( $LtA^a$ ) + unloaded time ( $UtA^a$ )), waiting time ( $WtA^a$ ), idle time ( $ItA^a$ ), and its efficiency ( $EA^a$ ) that are calculated by equations 3.21 to 3.27.

$$UtA^a = \sum_{\substack{j,i \\ T_{ji}^a \in T^a}} UT_{ji}^a \quad (3.21)$$

Equation 3.21 calculates all the time that AGV number  $a$  has been travelling without load, which it could be traveling to load the part/product from home/another machine or returning home for charging, during the process of all jobs.

$$LtA^a = \sum_{\substack{j,i \\ T_{ji}^a \in T^a}} LT_{ji}^a \quad (3.22)$$

Equation 3.22 calculates all the time that AGV number  $a$  has been travelling loaded to deliver the part/product to the destination machine during the process of all jobs.

$$WtA^a = \sum_{\substack{j,i \\ T_{ji}^a \in T^a}} WT_{ji}^a \quad (3.23)$$

Equation 3.23 calculates all the time that AGV number  $a$  has been waiting (either when it is waiting loaded for the machine to be free to deliver the part to the machine or when it is waiting unloaded for the machine to finish its operation and pick-up the part for the next destination or home) during the process of all jobs.

$$ItA^a = MS - RtA^a - \overline{TuA^a} \quad (3.24)$$

Equation 3.24 is all the time which AGV number  $a$  is idle (either waiting to be dispatched or equation 3.23) excluding the charging time. Waiting time is a part of the idle time in this formula.

$$RtA^a = \sum_{\substack{j,i \\ T_{ji}^a \in T^a}} RT_{ji}^a \quad (3.25)$$

Equation 3.25 is all the time that AGV number  $a$  is running either loaded or unloaded (combination of loaded and unloaded time of AGV number  $a$ ).

$$RT_{ji}^a = LT_{ji}^a + UT_{ji}^a \quad (3.26)$$

Equation 3.26 calculates the running time of AGV number  $a$  for doing  $T_{ji}$ .

$$EA^a = \frac{RtA^a}{MS} \times 100 \quad (3.27)$$

Equation 3.27 calculates the AGVs' operation efficiency, which determines how much work AGV has done during the makespan.

### 3.4 Optimization Algorithms Developed for the Model

Four EAs (GA, PSO, and 2 different hybrids of GA and PSO so called HGP1 and HGP2) have been developed to optimize the AGV scheduling model. The algorithms performances were later evaluated and compared to one another.

### 3.4.1 Genetic Algorithm

GA is a search algorithm based on the mechanics of the natural selection process. The major steps of GA algorithm development according to the study condition are described. However, for a thorough review of the GA, readers are referred to publications of Beheshti and Shamsuddin (2013); Elsayed et al. (2014); Holland (1975); Joshi (2014), and Thakur et al. (2014). Flowchart of the developed GA for the model is shown in Figure 3.5, and the entailed steps towards building the GA in relation to the model are described next.

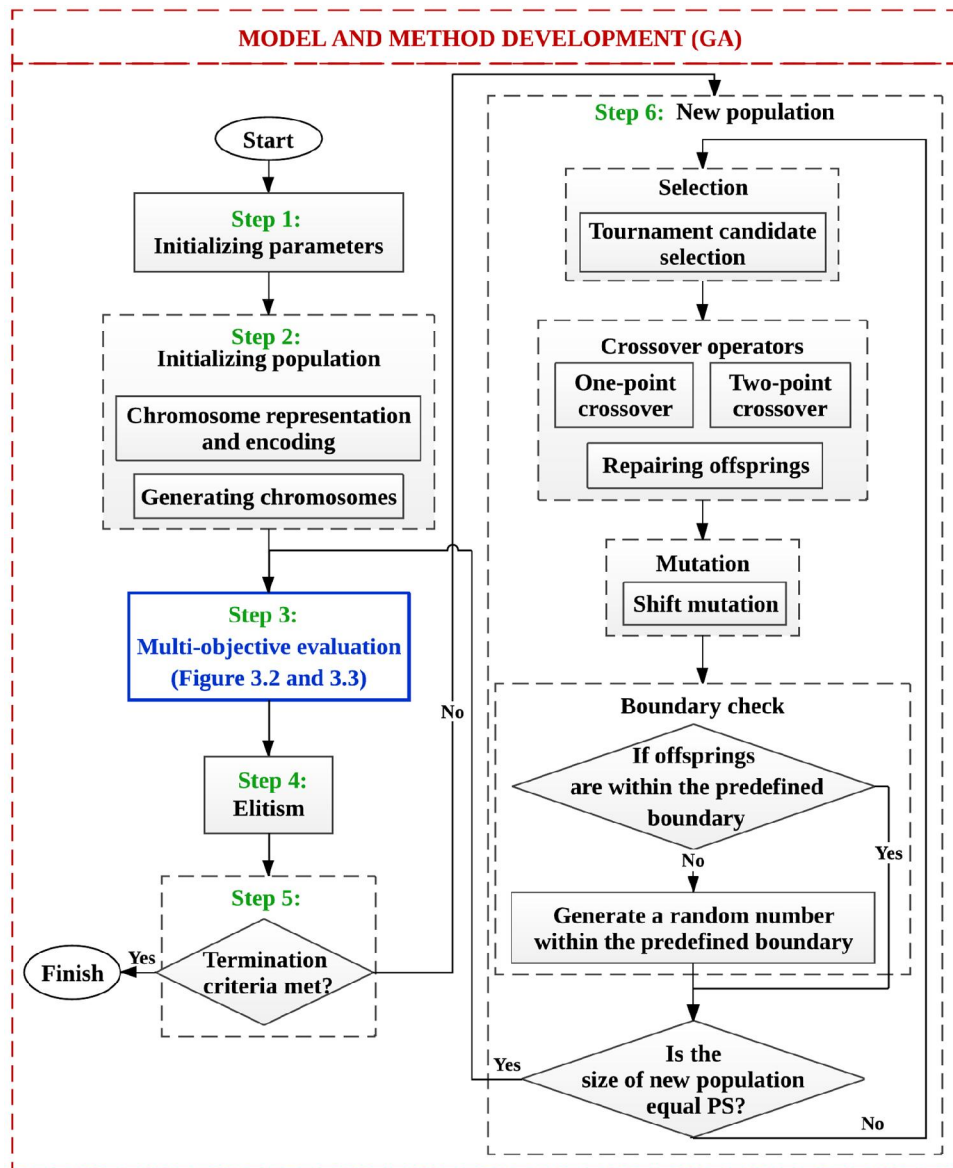


Figure 3.5: Flowchart of the GA

**Step 1. Initializing parameters.** It involves setting the parameters of the GA and creating the first generation of chromosomes. The general schematic of reading data for the problem is presented in Table 3.1. The first column shows a chromosome ( $C_r$ ) and the second one shows the genes ( $G_e$ ) of the chromosome. The encoding of each gene is presented in the third column, which will be discussed later.

Table 3.1: General schematic for reading data (Mousavi et al., 2017)

GA			PSO			Job ( $J_j$ )	Operation ( $O_{ji}$ )	Machine ( $M_{ji}$ )	Processing time ( $p_{ji}$ )
Chromosome ( $C_r$ )	Gene number ( $G_e$ )	Gene code ( $j$ )	Particle ( $PR_a$ )	Dimension Number ( $d$ )	Dimension code ( $j$ )				
$C_1$	$G_1$	1	$PR_1$	1	1	$J_1$	$O_{11}$	$M_{11}$	$p_{11}$
	$G_2$	1		2	1		$O_{12}$	$M_{12}$	$p_{12}$
	.	.		.	.		.	.	.
	.	.		.	.	.	.	.	.
	.	.		.	.	.	.	.	.
	$G_{m_1}$	1		$m_1$	1	$J_2$	$O_{1m_1}$	$M_{1m_1}$	$p_{1m_1}$
	.	2		.	2		$O_{21}$	$M_{21}$	$p_{21}$
	.	.		.	.	.	.	.	.
	.	.		.	.	.	.	.	.
	.	.		.	.	.	.	.	.
	.	2		.	2	$O_{2m_2}$	$M_{2m_2}$	$p_{2m_2}$	
	.	.		.	.	.	.	.	.
	.	.		.	.	.	.	.	.
	.	.		.	.	.	.	.	.
	.	$n$		.	$n$	$O_{n1}$	$M_{n1}$	$p_{n1}$	
	.	.		.	.	.	.	.	.
.	.	.	.	.	$J_n$	.	.	.	
.	.	.	.	.	.	.	.		
$G_\theta$	$n$	$\theta$	$\theta$	$n$	$O_{nm_n}$	$M_{nm_n}$	$p_{nm_n}$		

**Step 2. Initializing population.** A set of chromosomes is needed to create a population. For constructing a chromosome, it is necessary to define a proper genetic representation (encoding) due to its significant effects on all the subsequent steps of the GA.

**Chromosome representation and encoding.** As it was shown in Table 3.1, each chromosome is formed by genes. The order of genes represents the priority of operations, which decreases from left to right; and the genes' code defines the operations related to each job. Genes' codes are the same as their job number so that all the genes related to  $J_1$  operations have the code '1' and subsequently the code '2' is given to all the genes related to the operations of  $J_2$ , and so on. As the operations of each job are expected to be performed sequentially, the repetition of genes' code represents the corresponding operation number of the job as clearly described in the following example.

The number of genes in each chromosome equals the number of total operations in a job-set, which is expressed by

$$\theta = \sum_{j,j'=1}^{j,j'=n} m_{j,j'} \quad (3.29)$$

**Chromosome generating.** A chromosome ( $C_r$ ) is a random construct of operations, which is expressed by

$$C_r = \left\{ (m_{j,j'}) \mid j, j' = 1, 2, \dots, n \right\} = \left\{ \underbrace{(O_{11}, O_{12}, \dots, O_{1n})}_{m_1}, \underbrace{(O_{21}, O_{22}, \dots, O_{2n})}_{m_2}, \dots, \underbrace{(O_{n1}, O_{n2}, \dots, O_{nn})}_{m_j} \right\} \quad (3.30)$$

where  $j, j'$  are indexes of jobs,  $j, j' = 1, 2, \dots, n$  and  $m_{j,j'}$  = number of operations for each job.  $O_{ji}$  is the operation  $i$  of job  $j$ .

The process of generating and coding chromosomes is explained below via an example of 3 jobs ( $J_1, J_2$ , and  $J_3$ ). Each job has 4, 3, and 5 operations, respectively. Overall, there



are  $\theta = 4 + 3 + 5 = 12$  operations. Therefore, the chromosome is a random construct of

$\left[ \underbrace{1111}_4 \underbrace{222}_3 \underbrace{33333}_5 \right]$ . A sample could be  $[221132313133]$ . Here, code '1', '2', and '3'

imply operations of  $J_1$ ,  $J_2$ , and  $J_3$ , respectively. From the left, the first '2' represents the first operation of  $J_2$ , the second '2' represents the second operation of  $J_2$ , the first '1' represents the first operation of  $J_1$ , and so on.

**Step 3. Multi-objective evaluation.** After initializing the population size, each chromosome is evaluated with respect to the makespan and the number of AGVs utilized, while considering the battery charge of AGV, that are defined through equations 3.1 to 3.18. Then, the total fitness value will be calculated based on equations 3.19 and 3.20.

**Step 4. New population.** New population will be produced based on the below sub-steps: selection, crossover, elitism, and mutation operation.

**Selection.** To constantly enhance the overall fitness of the population, selection helps to discard the bad/weak chromosomes and only keep the best ones in the population. It increases the likelihood of selection of individuals with better fits for the next generation. There are a few different selection methods but their basis is the same. The tournament candidate selection, which is a proportionate random selection method suitable for multi-objective optimization, is used in this study (Shukla et al., 2015).

In this method, every individual in the population is paired at random with another. The fitness values of each pair are compared. The fitter individual of the pair moves on to the next round, while the other is disqualified. This continues until there are a number of winners which is equal to the desired number of parents. Then, this last group of winners is paired as the parents for new individuals (Chudasama et al., 2011).

**Crossover.** Crossover operator generates two new chromosomes for the next generation out of two selected chromosomes by exchanging some of their genes. This study employed two crossover operators based on partial strings exchange; a one-point crossover and a two-point crossover (Spears & Anand, 1991). The one-point crossover randomly divides the two parent chromosomes into two substrings and two new chromosomes are obtained by exchanging the second substring and maintaining the first one or vice versa. It is illustrated in Figure 3.6 based on the example in step 2.

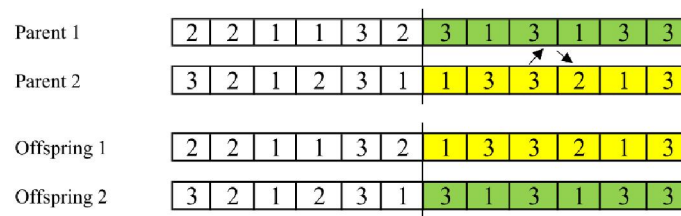


Figure 3.6: Example of one-point crossover (Mousavi et al., 2017)

The two-point crossover is similar to the one-point crossover and is illustrated in Figure 3.7. Two cut points are randomly chosen and three substrings are determined for each parent. The first and the third pair of substrings are exchanged and two new chromosomes are created.

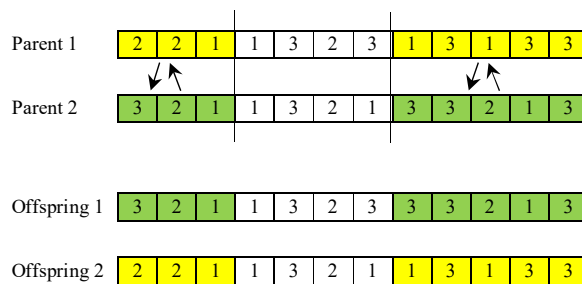


Figure 3.7: Example of two-point crossover

The offspring of crossover between the strings may not produce a legal encoding, for example, uncorrected number of operations per job may be seen. Therefore, they should be repaired and legalized. For repairing mechanism, counting from the left, the redundant genes will be deleted and the missing ones would be replaced; thus, each offspring can

comprise all the operations of all the jobs. Repair mechanism is shown in Figures 3.8 and 3.9.

Legal chromosomes for the example in step 2 should include four codes '1', three codes '2', and five codes '3'. In Figure 3.8a, counting from the left, in offspring 1, code '2' is repeated four times, but there are three operations for  $J_2$ , so it should repeat three times. There is one code '2' that is redundant and should be replaced by the missing code. Code '1' is repeated four times, which is correct, but code '3' is repeated only four times, which should be five times. So, in Figure 3.8b, the fourth code '2' will be replaced by number '3'. In offspring 2, number '2' is repeated two times and number '3' is repeated six times, so the last code '3' will be changed to code '2'.

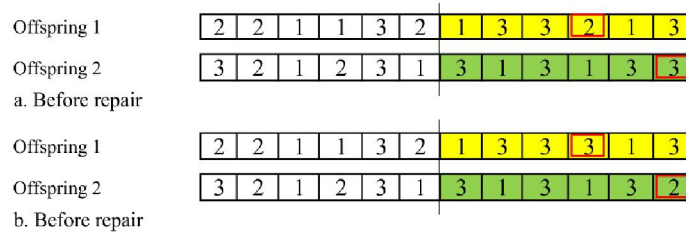


Figure 3.8: Repairing offsprings out of one-point crossover (Mousavi et al., 2017)

The same procedure would be applied to repair two-point crossover (Figure 3.9).

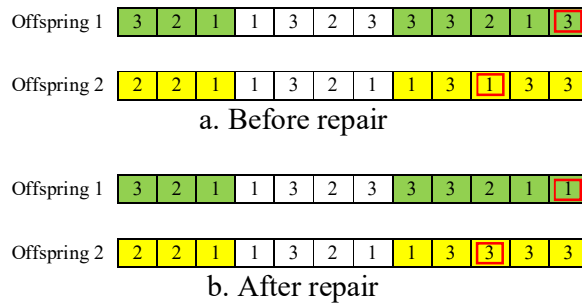


Figure 3.9: An example of repairing offsprings out of two-point crossover

The number of crossovers is calculated based on the crossover rate ( $CR$ ) and population size ( $PS$ ) using

$$\text{Number of crossovers} \cong \frac{(CR \times PS)}{2} \quad (3.31)$$

**Mutation.** Mutation is another important operator of GA that initiates extra variability in a population to create and maintain the diversity. The number of mutations in each generation is calculated using equation 3.32 based on the mutation rate ( $Pm$ ), population size ( $PS$ ), and maximum gene code ( $G_{max}$ ).

$$\text{Number of mutations} \cong (PS \times G_{max}) \times Pm \quad (3.32)$$

Of the several mutation types, shift mutation is used in this study (Nearchou, 2004). In shift mutation, a gene is selected randomly as a move-point and inserted in a random position (inserting point) as it is shown in Figure 3.10. Based on the coding used in this study, chromosomes produced out of shift mutation are legal and no need to be repaired.

Before mutation	3	2	1	2	3	1	1	3	3	2	1	3
After mutation	3	2	2	3	1	1	3	3	1	2	1	3

Figure 3.10: Example of shift mutation operator (Mousavi et al., 2017)

**Boundary check:** In boundary-constrained problems, the parameter values of the trial vectors need to be checked whether they lie within the range or not. In case they violate the boundary constraint they should be adjusted.

**Elitism.** The first three best chromosomes from each generation are transferred directly to the next generation in the elitism step to avoid annihilation. It is possible to maintain a fixed fitness value in some generations, but elitism makes sure they will never deteriorate.

**Step 5. Termination.** The loop of chromosome generations is terminated when certain conditions are met. The conditions are; either the number of generations reaches its maximum or there are no changes or marginal changes in the elite solution. The elite chromosome is returned as the best solution once the termination criteria are met.

### 3.4.2 Particle Swarm Optimization

PSO is a population based stochastic technique inspired by social behaviour of bird flocking or fish schooling. Extensive reviews on PSO algorithm development can be found in Du et al. (2015); Gao et al. (2015); Kennedy et al. (2001); Li and Yao (2012), and (Song, 2014). The developed PSO for the model is shown in Figure 3.11. The PSO configuration for the model is described in details in the following steps:

**Step 1. Initializing parameters.** Initialization involves setting the parameters of the PSO to create a group of particles to make the initial swarm in the next step. The general scheme for reading the data in the problem is presented in Table 3.1. The forth column shows a particle ( $PR_a$ ) and the fifth one shows dimensions of the particle ( $d$ ). The dimensions' codes are presented in the sixth column, which will be discussed later.

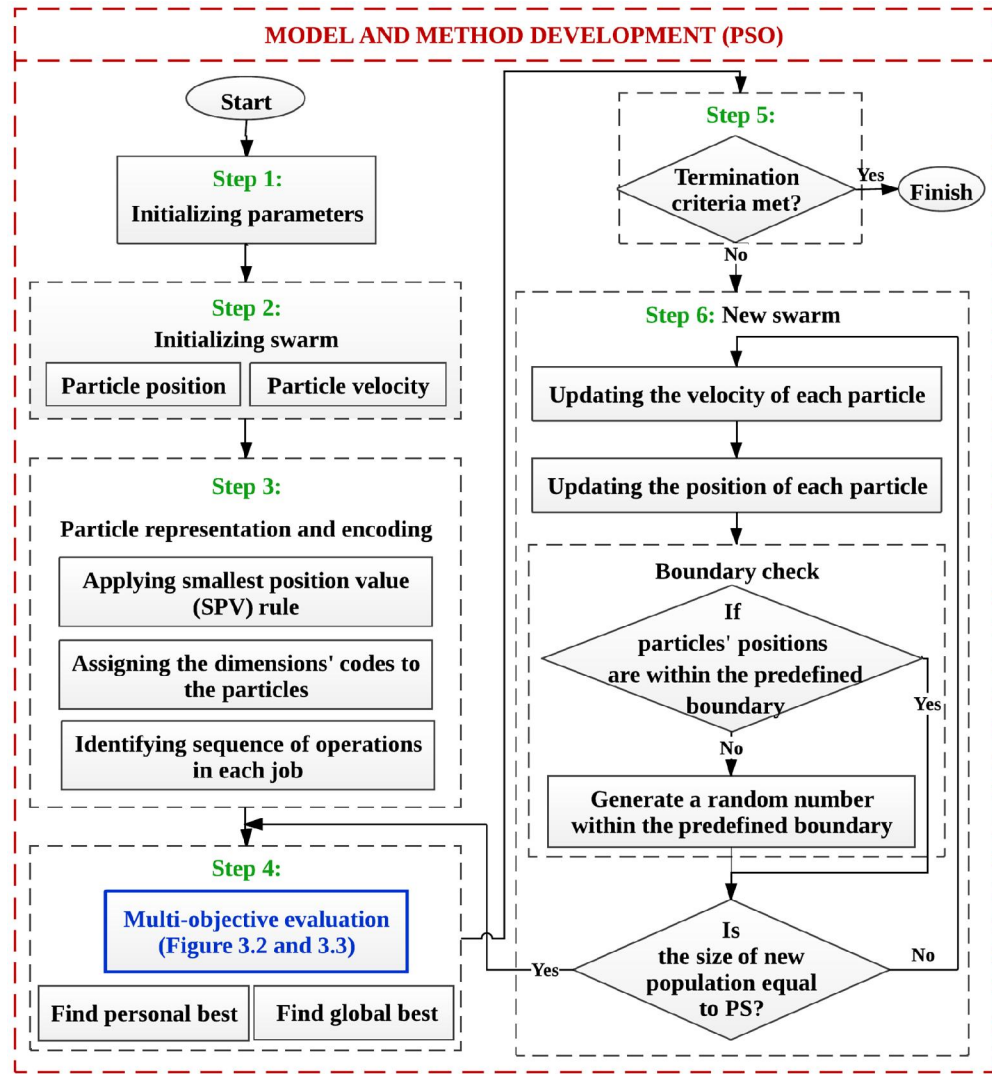


Figure 3.11: Flowchart of PSO

**Step 2. Initializing population (swarm).** A group of particles are needed to create a swarm. Each particle has position ( $Q$ ) and velocity ( $V$ ) in the search space at iteration ( $t$ ), where they are described briefly in the following sub-steps:

**Particle position.**  $Q_{\alpha}^t$  denotes the  $\alpha^{th}$  particle in the swarm at iteration ' $t$ ' and is represented by ' $\theta$ ' dimensions as

$$Q_{\alpha}^t = [q_{\alpha 1}^t, q_{\alpha 2}^t, q_{\alpha 3}^t, \dots, q_{\alpha \theta}^t] \quad (3.33)$$

where  $q_{\alpha d}^t$  is the position value of  $\alpha^{th}$  particle with respect to  $d^{th}$  dimension and  $d=1, \dots, \theta$ . First position of particle is filled by two-digit numbers for ' $d$ ' dimensions of

the particle using equation (3.34). The number of dimensions is equal to the total number of operations, which is calculated by equation (3.29).

$$q_{\alpha d}^0 = q_{\min} + (q_{\max} - q_{\min}) \times \varphi_1 \quad (3.34)$$

where  $q_{\min} = 0$ ,  $q_{\max} = 10$  and  $\varphi_1$  is a uniform random number between 0 and 1.

**Particle velocity.** Initial velocities for the PSO particles are generated by the below formula:  $V_{\alpha}^t$  denotes the velocity of  $\alpha^{th}$  particle at iteration ( $t$ ). It can be identified by

$$V_{\alpha}^t = [v'_{\alpha 1}, v'_{\alpha 2}, v'_{\alpha 3}, \dots, v'_{\alpha \theta}] \quad (3.35)$$

where  $v'_{\alpha d}$  is the velocity of  $\alpha^{th}$  particle with respect to  $d^{th}$  dimension. Velocity should be set within the limits so that it will not overshoot the search space. Initial velocities for the PSO particles are generated by the formula below:

$$v_{\alpha d}^0 = v_{\min} + (v_{\max} - v_{\min}) \times \varphi_2 \quad (3.36)$$

where  $v_{\min} = 0$ ,  $v_{\max} = 10$  and  $\varphi_2$  is a uniform random number between 0 and 1.

**Step 3. Particle representation and encoding.** Every possible sequence of operations is considered as a particle, where the dimension of the particle represents each operation. Three sub-steps for encoding a particle are as follows: applying smallest position value (SPV) rule, assigning the dimensions' codes to the particles, and identifying sequence of operations in each job.

**Applying smallest position value (SPV) rule.** SPV is a rule that facilitates transformation of the continuous PSO algorithm to discrete cases applicable to all types of the scheduling problems (Tasgetiren et al., 2004). As an example, for better understanding of SPV rule, the corresponding sequence of a given continuous position like [0.3, 1.2, 0.9, 2.4] would be [4, 2, 3, 1]. In a descending order, '0.3' is the smallest value and its sequence will be '4'; '2.4' is the largest so its order in the group will be '1'.

**Assigning the dimensions' codes to the particles.** In this stage, the dimensions' codes as it is shown in the 6<sup>th</sup> column of Table 3.1 are assigned to the particles. Dimensions' codes are based on the job number.

**Identifying sequence of operations in each job.** From the left side, the first appearance of a job number is assumed the first operation of that job (i.e.,  $O_{j1}$ ). Similarly, the second-time repetition of the same job number stands for the second operation of the same job (i.e.,  $O_{j2}$ ) and so on. Once the first encountered generated number is assigned to the first operation of a job, this technique automatically handles the precedence constraints.

Table 3.2: Encoding of a sample particle (Mousavi et al., 2017)

Particle sample	0.2	0.37	0.17	0.51	0.73	0.42	0.93	0.35	0.69	0.84	0.65	0.05
Applying SPV rule (giving the numbers from one based on ascending order)	3	5	2	7	10	6	12	4	9	11	8	1
Assigning the dimensions' codes to the particles	1	2	1	2	3	2	3	1	3	3	3	1
Identifying sequence of operations in each job	$O_{11}$	$O_{21}$	$O_{12}$	$O_{22}$	$O_{31}$	$O_{23}$	$O_{32}$	$O_{13}$	$O_{33}$	$O_{34}$	$O_{35}$	$O_{14}$

The stages of encoding an example with 3 jobs are shown in Table 3.2. Each job has 4, 3, and 5 operations respectively. The total number of operations is 12, which means the particle sample will have 12 dimensions, and each dimension being randomly generated using equation 3.29 and shown in the first row of Table 3.2. In the second row of the table, based on SPV rule, the numbers of 1 to 12 are assigned to the particles in an ascending order. In the third row, the dimensions' code based on the job numbers are given to the particles as follows: first four numbers are assigned to the first job, so their code is '1', followed by the second three numbers assigned to the second job, so their code is '2' and the remaining five numbers are assigned to the third job and their code is '3'. The sequence of operations in each job is shown in the fourth row of the Table 3.2. From the left, the first particle has the code '1', so it belongs to job 1 and it is the first code '1', which makes it the first operation of job 1 denoted by  $O_{11}$ ; the next code is '2',



so it belongs to job 2, but as it is the first code '2', it is the first operation of job 2. The same structure is followed for the remained 10 operations.

**Step 4. Multi-objective evaluation.** Once the swarm is generated, each particle is evaluated with respect to the obtained makespan and number of AGVs, while considering the AGV battery charge, which are defined through equations 3.1 to 3.18. Then, the total fitness value will be calculated based on equation 3.19 and 3.20.

**Personal best.**  $B_{\alpha}$  represents the best position associated with the best permutation and fitness value of the particle  $\alpha$  obtained so far and is called the personal best. For each particle,  $B_{\alpha}$  is determined and updated at each iteration.

**Global best.**  $G'$  denotes the best position of the globally best particle achieved so far in the whole swarm.

**Step 5. New swarm.** To produce a new swarm, the position and velocity of the particles should be updated. Updated particles will be evaluated again according to the step four and their best local and global particle will be determined. This procedure will be repeated up to a point where the termination criterion is satisfied. The updating procedure is explained as follows:

**Updating the velocity of each particle.** The velocity of each particle is updated using

$$v_{\alpha d}^{t+1} = \omega v_{\alpha d}^t + C_1 \varphi_1 (B_{\alpha d}^t - q_{\alpha d}^t) + C_2 \varphi_2 (G_d^t - q_{\alpha d}^t) \quad (3.37)$$

where  $v_{\alpha d}^t$  and  $v_{\alpha d}^{t+1}$  are the velocity of  $\alpha^{th}$  particle on  $d^{th}$  dimension at instance  $(t)$  and  $(t+1)$ , respectively;  $q_{\alpha d}^t$  and  $q_{\alpha d}^{t+1}$  are the positions of  $\alpha^{th}$  particle on  $d^{th}$  dimension at instance  $(t)$  and  $(t+1)$ , respectively.  $t$  is the previous iteration,  $d$  is the dimension and  $\alpha$  is

the index for particles,  $\alpha=1, \dots, S^t$ ,  $S^t$  is swarm size at iteration ( $t$ ),  $\varphi_1$  and  $\varphi_2$  are uniformly distributed random numbers in the interval of  $[0, 1]$ .  $C_1$  is self-confidence while  $C_2$  is swarm confidence and their values should be tuned based on the experiment, and  $\omega$  is the inertia weight parameter.

**$C_1$  (self-confidence) and  $C_2$  (swarm confidence).** Since  $C_1$  helps in self-exploration (or experience) of a particle, it can be treated as self-confidence coefficient of the particle. Similarly, it is appropriate to treat  $C_2$  as swarm confidence, since it is the coefficient, which contributes in moving the particle toward the global best direction by considering the motion of all the other particles in the swarm in the preceding program iterations.  $C_1$  and  $C_2$  are sometimes also known as cognitive and social parameters respectively (more details in “tuning the parameters” section).

**Inertia weight ( $\omega$ ).** Inertia weight is a parameter to control the impact of the previous velocity on the current velocity (Kuo et al., 2009; Xia & Wu, 2005). It is an important parameter to be optimized for obtaining better results in PSO. A large inertia weight facilitates searching new areas while a small weight facilitates fine searching in the current search space. To strike a balance between global exploration and local exploitation, a suitable selection of inertia weight is necessary (more details in “Tuning the parameters” section).

**Updating the position of each particle.** The position of particle is updated using the updated velocity as below:

$$q_{\alpha d}^{t+1} = q_{\alpha d}^t + v_{\alpha d}^{t+1} \quad (3.38)$$

Once the positions and velocities for the next instance are calculated, they will be checked whether they are in the prescribed limits or not. These limits for positions and velocities are  $[0, Q_{max}]$  and  $[-V_{max}, V_{max}]$  respectively.

**Step 6. Termination.** The loop of swarm groups is terminated when certain conditions are met. The conditions are; either the number of iterations reaches its maximum or there are no changes or marginal changes in global best evaluation of particle. The particle with global best is returned as the best solution once the termination criteria are met.

### **3.4.3 Hybrid GA and PSO**

Taking advantage of compensatory properties of PSO and GA, two hybrids of GA and PSO (so called HGP1 and HGP2) were developed for the problem. PSO robustness, independency from the problem, and social interaction knowledge along with GA advantage of evaluating each individual and choosing the better ones are the characteristics incorporated into a hybrid algorithm. The development procedure of both hybrids and their flowcharts are described in the following sections.

#### **3.4.3.1 The First Hybrid GA-PSO (HGP1)**

In the proposed HGP1, the initial swarm was created and evaluated; the positions and velocities of the particles were updated by using the concerned equations. Next, in order to effectively exploit the search space, the well-known genetic operators (selection, crossover, and mutation) were employed. Selection (the tournament as in GA in section 3.4.3) finds parents from the updated particles for crossover and mutation step. The crossover operation has been used in the GA segment to avoid premature convergence; and mutation operation was applied to maintain the diversity of the swarms. The operators' concept hired from GA was incorporated into PSO and the resulting hybrid PSO was named HGP1. Figure 3.12 illustrates the steps of HGP1.

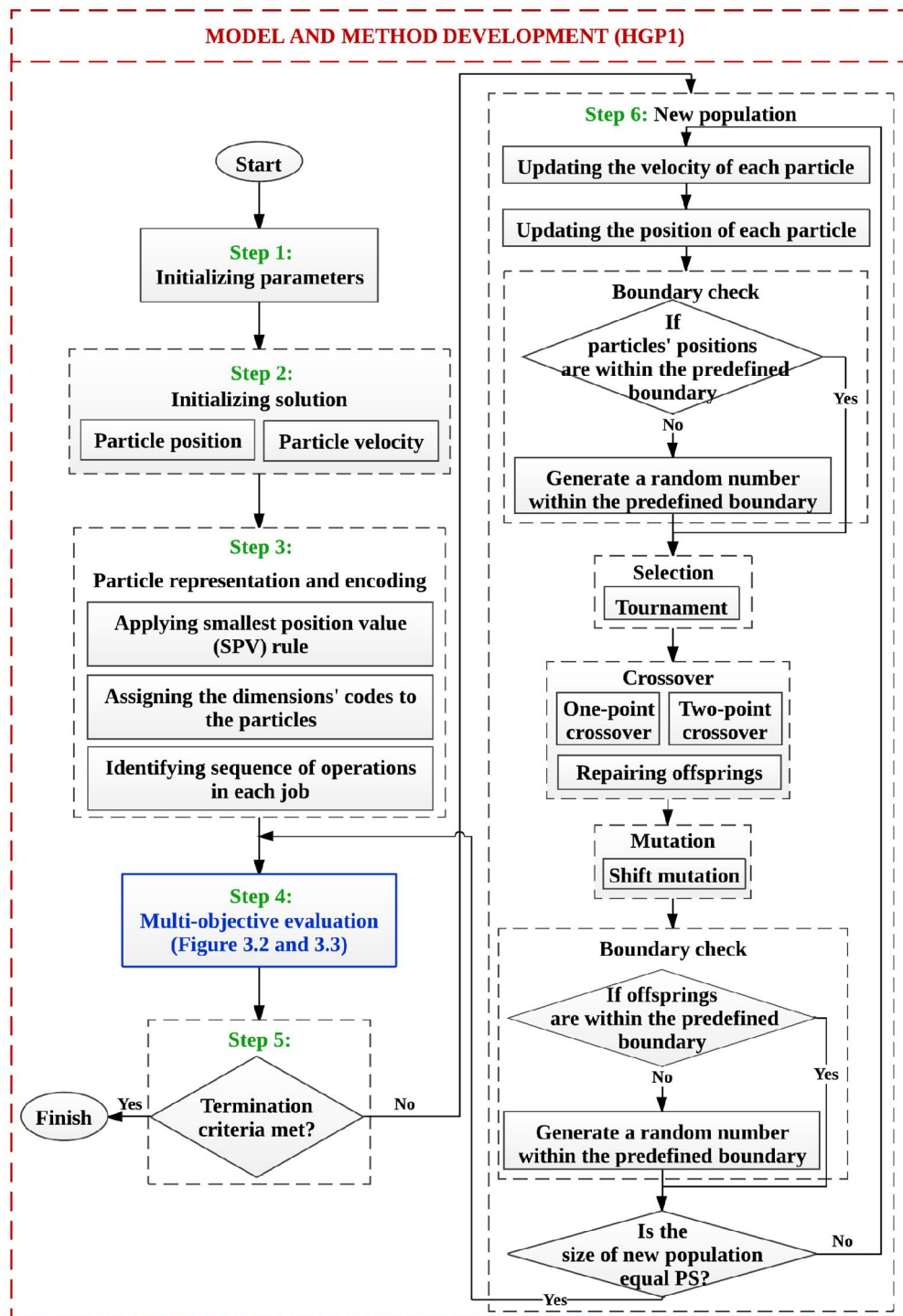


Figure 3.12: Flowchart of HGP1

### 3.4.3.2 The Second Hybrid GA-PSO (HGP2)

Figure 3.13 illustrates the steps followed in HGP2 and it is briefly explained in the following paragraphs. In the second hybrid (HGP2), initialization and encoding, and evaluation steps were performed the same as in PSO.

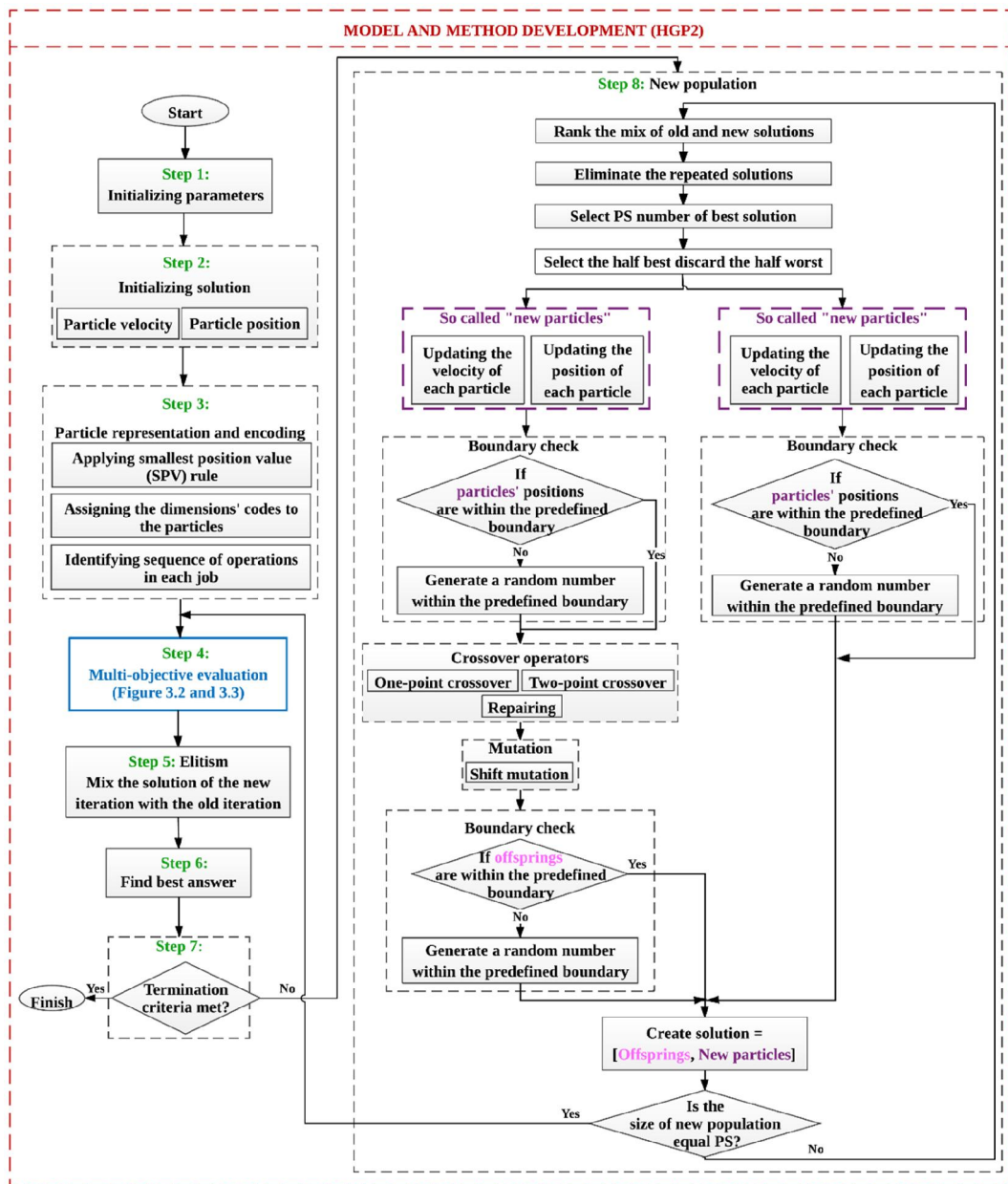


Figure 3.13: Flowchart of HGP2

Next, elitism was applied in a different way than the one used in GA. In this step, obtained results were mixed with the results from the previous iteration to prevent good results from fading away. In the selection step, repetitive solutions of the two generations were eliminated and from the remaining particles a swarm inclusive of the best particles was extracted. Then again, particles were ordered and the best half was retained and the other half was discarded. Next, to create a population with the required size; in two parallel segments, once both the GA and PSO operators were applied to the retained best particles

of the last step and once only the PSO operators were applied. The output of each segment would construct one half of the new population size. Therefore, a swarm conforming to the required population size was finally created from the best offsprings and particles.

In each of the segments, the position and velocity of the half-size population were updated and checked if they were within the predefined boundary. So that one half of the new population is ready up to this stage. For the next segment, to create the other half of the new population, GA operators (crossover and mutation) were also applied and reviewed in terms of the boundary condition accordance. In both the segments, if the boundary conditions were not met, algorithm would replace the outliers with random numbers within the predefined boundaries and proceeds until termination criteria are met, then the best answer in step 6 would be the fitness value obtained.

### **Tuning the Parameters**

Tuning the metaheuristics' parameters, due to their impact on the performances obtained, is of paramount importance. The parameters are different in each algorithm. The four EAs' performances depend on parameters like population size, inertia factor ( $\omega$ ), self-confidence ( $C_1$ ), swarm confidence ( $C_2$ ), crossover, and mutation rate. Regarding population size, small populations do not provide enough diversity among the individuals. Increasing the population size also does not necessarily improve the performance as it can be observed from the multimodal functions, but increases the diversity and consequently increases the computation time for each generation. The same holds for the number of generations to be simulated (Andersson et al., 2016; Eiben & Smit, 2011a; Karafotias et al., 2015). Thus, increase in crossover rate increases the opportunity for recombination but also disruption of good combinations. Increase in mutation rate escalates the random search capability and facilitates introduction of a new gene or re-introduction of the lost genes (Veček et al., 2016). Guo and Yang (2011), in this regard,

have also stated that to avoid local optimal solutions; individual rate of mutation and the hereditary generations of population can be increased. Of the available tuning methods for evolutionary algorithms, application of the Brute-force technique in cases with less than 10 parameters has been advised (Silberholz & Golden, 2010). Brute-force technique involves testing of  $m$  parameter values for each of the  $n$  parameters, a procedure that should test  $n^m$  configurations over a subset of the problem instances (Palonen et al., 2009; Sinha et al., 2014; Veček et al., 2016).

Values of  $C_1$  and  $C_2$  vary in different studies and the range is between 0.1 and 2. However, values of below 1 are more frequently observed in different studies (Karafotias et al., 2015). The inertia weight can be tuned in two ways. In the first case, its value is changed from 0.1 to 0.9 and every time the effect is monitored (Lobo et al., 2007). The other approach is the linear decreasing inertia weight (LDIW) which  $\omega$  would be varied with time and calculated by equation 3.39 (Andersson et al., 2016; Eiben & Smit, 2011b). In this way, the algorithm could have more global search ability at the beginning of the run while having the more local search ability near the end of the run, because the inertia factor moves from a large value ( $\omega_{max}$ ) to a relatively small value ( $\omega_{min}$ ) during the run.

$$\omega = \omega_{max} - It \times \frac{\omega_{max} - \omega_{min}}{It_{max}} \quad (3.39)$$

where  $It$  is the current iteration number,  $It_{max}$  is the maximum iterations during the evolutionary process,  $\omega_{max}$  and  $\omega_{min}$  are maximum and minimum values of  $\omega$ , respectively.

### 3.5 Programming in MATLAB

There are different software for programming of EAs such as C/C++, Java, MATLAB, etc. Yu and Gen (2010b), and (Vasava, 2014) have suggested MATLAB as a

programming environment for implementing EAs. It allows to import data in .xls, .csv files, etc. It also has powerful plotting tools for easier data visualization. MATLAB (matrix laboratory) is a proprietary programming language developed by MathWorks Ltd which is a multi-paradigm numerical computing environment and fourth-generation programming language. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, and creation of user interfaces. It has the ability to interface with programs written in other languages, including C, C++, C#, Java, Fortran, and Python (The MathWorks, 2016). To run MATLAB, the computer requirements are any Intel or AMD x86-64 processor, Windows Server 2008 R2 and higher or Windows 7 and higher, 4 GB RAM with Simulink, and 2 GB without, hardware accelerated graphics card supporting “Open graphics library (OpenGL)” 3.3 with 1GB “Graphics processing unit (GPU)” memory (The MathWorks, 2016). The model and algorithms of this study have been programmed in MATLAB software version 2014a (64 bit). The programming codes are at the appendix.

### **3.6 Evaluation of Model and Algorithms**

With every metaheuristic having a unique functionality, their inter-comparison is in many ways more difficult than other algorithmic comparisons. Using testbed should be the first consideration when comparing two metaheuristics. Two kinds of testbeds can be used (Silberholz & Golden, 2010): (1) using the existing testbeds, and (2) developing new testbeds. When comparing a new metaheuristic to existing ones, it is better to test it on the problem instances already tested at previous studies. There are many cases where this is either insufficient or not possible. For instance, when writing a metaheuristic for a new problem or model, there will be no prior-testbed for that problem, thus it is necessary to develop a new testbed corresponding to the problem specifications.

New testbeds should represent real-life cases with various sizes and difficulty levels. One of the key points in the testing of metaheuristics is the testbed size that must be a large



problem (Silberholz & Golden, 2010). In the same vein, Jans and Degraeve (2007) mentioned that algorithms can perform differently regarding either runtime or quality of solution on large-sized instances. It is also advised by Gen and Lin (2012) to test the performance of algorithms by conducting large-sized problems. Problems with more than 100 operations are often categorized as large-sized problems (Eren & Güner, 2007; Özgüven et al., 2010). In developing new testbeds, providing accessibility to the testbed should be considered to let other researchers use them and make comparisons for future studies. One effective way to make this happen is to create a simple function for the testbed generation. Publishing them is another way to make them accessible (Silberholz & Golden, 2010).

With the above background in mind and having a new criterion (AGVs' battery charge considerations) included in the model that distinguishes it from existing models, it was realized that the available testbeds are not suitable to the problem at hand both in terms of size and the problem definitions. Therefore, four new testbeds were developed in different sizes (small, medium, and large) with different machines, operations and job numbers to test the model functionality and assess the model response to the problem size.

### **3.7 Validation**

This section expresses validation of the model and validation of the optimization result.

#### **3.7.1 Model Validation**

Validation step determines if the model is a satisfying image of the real system. It indicates that the system performance is reliable enough to be applied to the real-world cases and satisfy the model objectives (Aydemir & Koruca, 2015; Hillston, 2003). To validate the model, it has been tested on benchmark problems which are considered for this purpose. The proposed model has been validated through the well-known Bilge and

Ulsuy (Bilge & Ulusoy, 1995) problems with data of ten job sets on two layouts. Then, the results from previous algorithms which had used the same data for scheduling are used for comparison purpose.

### **3.7.2 Validation of Optimization Results**

Computer simulation is a common tool applied to comprehend complex systems and assess their practicality (Azimi, 2011; Ba et al., 2016; Kelton et al., 2004; Sinha et al., 2014). In simulation approach, a computerized model is designed to simulate the system's operations and configurations so that different testbeds can be conducted to evaluate the model performance and practicality accordingly (Keesman, 2011; Li et al., 2015). Therefore, a simulation practice in FlexSim computer simulation software (version 16.2.2) has been performed.

FlexSim Software Products Inc. (FSP) creates simulation software and provides simulation modeling services. It introduced FlexSim simulation software (FlexSim) 1.0 in February 2003. It brought a new modern simulation engine, a smooth integration with C++, and a three-dimensional (3D) modeling environment. From the beginning, FlexSim had standard features of discrete event simulation packages. The least specification of a computer running FlexSim is any modern Intel or AMD processor, 4 GB RAM, Windows Vista or higher, any GPU that supports OpenGL 3.1 or higher, and ".NET Framework" is an essential additional software. Through this software, one representative medium-sized testbed (testbed 2) was simulated and evaluated in terms of the whole system performance. The system analysis was performed based on the statistics and graphical outputs of the simulation step for both the AGVs and machines (FlexSim Software Products, 2016).

### **3.8 Summary**

The scheduling model derivation fundamentals were explained according to the three criteria, which are classified into two main classes of makespan minimization and number of AGVs minimization with AGVs' battery charge consideration. Optimization of the scheduling model was performed using the four evolutionary algorithms of GA, PSO, and two different hybrids of GA and PSO named HGP1 and HGP2. Configurations of the four algorithms were illustrated in this chapter and it was discussed why the hybrid algorithm can be an improvement over its constituent algorithms. The weighted sum method was used in the model to acquire a single aggregated value from the different objectives applied. Next, to investigate the effect of optimization on AGV specification/behavior, some of the AGVs' specifications in the system were computed and compared with results of the model before optimization. The model and algorithms of this study have been programmed in MATLAB software version 2014a. For validating model, it has been tested on some well-known benchmark problems. Final, simulation by FlexSim software was used to validate the optimization result.

## **CHAPTER 4: RESULTS AND DISCUSSION**

### **4.1 Introduction**

Having developed the proposed scheduling model and the four EAs (PSO, GA, HGP1, and HGP2) as explained in chapter three, they have been applied to several testbeds to evaluate their functionalities. The algorithms' performances at any of the testbeds are shown and discussed in this chapter. The algorithms' performances were assessed and compared with one another in terms of their worst, mean, and best results, computational time, standard deviation, and convergence rate. Next, effect of the optimization process on AGVs' specifications/behaviors such as running time (loaded and unloaded), idle time, and AGVs' operation efficiency is discussed. In later sections of this chapter, the simulation results obtained using FlexSim software are presented to elaborate upon the proposed model validity.

### **4.2 The Developed Model**

To fulfill the first objective of the study, development of the optimization model for AGV scheduling was pursued by reviewing the literature and defining a set of criteria that can potentially result in a practical model conforming to the real environment situation. To have a smooth flow of information in the dissertation, the conceptual basis and mathematical aspects of the developed model were described in the methodology chapter (refer to Figures 3.2 and 3.3). Based on the selected scheduling criteria, model was developed and translated into algebraic expressions applicable to the optimization algorithms running in MATLAB programming software. The corresponding program codes are shown in the appendix. Assessment of the model applicability was examined using four testbeds and the results are expanded in this chapter.

### 4.3 Evolutionary Algorithms

To achieve the second objective of the study, four evolutionary algorithms of GA, PSO, and two hybrids of GA-PSO (HGP1 and HGP2) were developed for the model. The evolutionary algorithms as described in the previous chapter were coded in MATLAB, and the experiments were run on a desktop computer with a 2.80 GHz processor and 4 GB RAM. The program codes of each algorithm are placed in the appendix. The algorithms performance at every testbed is discussed in later sections.

### 4.4 Model and Algorithms' Performance Evaluation

Four new testbeds with different sizes (small, medium, and large) in terms of the number of operations, jobs, and machines were defined to provide a comprehensive comparison basis for assessment of the model and algorithms' practicality and performance.

#### 4.4.1 Parameter Setting of the Algorithms

In addition to the EAs' structural development, obtaining the optimal parameter setting of each EA was the next step toward accomplishing the second objective and preparing the EAs. To obtain the best setting in any of the optimization algorithms, a series of trial experiments based on Brute-force method was performed to estimate the optimal parameters. Each algorithm has been run with different population sizes and different iteration numbers using different settings of crossover and mutation rates,  $C_1$ , and  $C_2$  as shown in Table 4.1. Based on the experimental approach followed, all the four algorithms obtained their best results at a run with the population size and iteration number of 100. For GA, the optimal rates of crossover and mutation were found to be 0.2, and 0.03 respectively. For PSO, optimal number for  $C_1$  and  $C_2$  were found to be 2. As LDIW was used in the study, it is recommended for the  $\omega_{\min}$  to be 0.4, and  $\omega_{\max}$  to be 0.9 for all algorithms including  $\omega$  (Kessentini & Barchiesi, 2015; Shi & Eberhart, 1999). For HGP1, the parameters were found to be the crossover and mutation rates of 0.3 and 0.05

respectively,  $C_1=0.02$ , and  $C_2=1$ . For HGP2, crossover and mutation rates of 0.2 and 0.08 respectively, and  $C_1=0.01$ ,  $C_2=0.9$ .

Table 4.1: Different settings of parameters experimented

	HGP1 and HGP2	GA	PSO
Run number	20		
Population size	20, 50, 100, 150, 200		
Generation (iteration) number	20, 50, 100, 150, 200		
Crossover rate	(0.01,...,0.09, 0.1,...0.9, 1)	NA	NA
Mutation rate	(0.01,...,0.09, 0.1,...0.9, 1)		
C1	(0.01,...,0.09, 0.1,...0.9, 1, 1.01,...1.09,...1.1, 1.9, 2)	NA	(0.01,...,0.09, 0.1,...0.9, 1, 1.01,...1.09,...1.1, 1.9, 2)
C2	(0.01,...,0.09, 0.1,...0.9, 1, 1.01,...1.09,...1.1, 1.9, 2)		(0.01,...,0.09, 0.1,...0.9, 1, 1.01,...1.09,...1.1, 1.9, 2)

Any alteration of the above parameters' value may lead to convergence at higher (worse) results than while using the optimal parameters. Therefore, the obtained optimal parameters' settings were used in the developed EAs for the model at any of the testbeds. Outcomes are presented in this section categorized into four subsections relating to each testbed.

#### 4.4.2 Performance at Testbed 1

The first example was a small testbed inclusive of 6 jobs ( $J_1, \dots, J_6$ ) processing on 6 machines ( $M_1, \dots, M_6$ ), and each job with 2 to 5 operations (total of 19 operations). Figure 4.1 shows the layout of testbed 1 with the routes being one-way. The AGV travel time among L/U point and machines is shown in Table 4.2. Next, Table 4.3 demonstrates the processing time of every operation on the machines for this testbed.

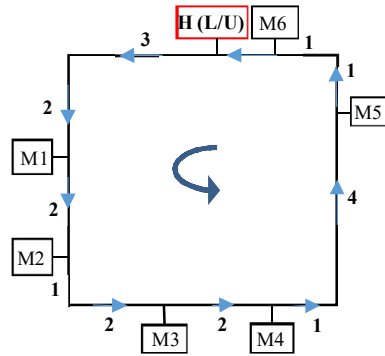


Figure 4.1: The layout of testbed 1

Table 4.2: AGV travel time (minutes) among L/U point and machines

	L/U	M <sub>1</sub>	M <sub>2</sub>	M <sub>3</sub>	M <sub>4</sub>	M <sub>5</sub>	M <sub>6</sub>
L/U	0	5	7	10	12	17	19
M <sub>1</sub>	15	0	2	5	7	12	14
M <sub>2</sub>	13	18	0	3	5	10	12
M <sub>3</sub>	10	15	17	0	2	7	9
M <sub>4</sub>	8	13	15	18	0	5	7
M <sub>5</sub>	3	8	10	13	15	0	2
M <sub>6</sub>	1	6	8	11	13	18	0

Table 4.3: The processing time (minutes) of every operation on the machines

Job	1	1	1	1	2	2	2	3	3	3	3	3	4	4	4	5	5	6	6
Operation	1	2	3	4	1	2	3	1	2	3	4	5	1	2	3	1	2	1	2
Machine	M <sub>2</sub>	M <sub>4</sub>	M <sub>5</sub>	M <sub>6</sub>	M <sub>3</sub>	M <sub>4</sub>	M <sub>6</sub>	M <sub>1</sub>	M <sub>2</sub>	M <sub>3</sub>	M <sub>1</sub>	M <sub>4</sub>	M <sub>6</sub>	M <sub>4</sub>	M <sub>5</sub>	M <sub>5</sub>	M <sub>1</sub>	M <sub>1</sub>	M <sub>2</sub>
Operation time	30	21	24	27	15	24	13	16	21	18	14	25	25	19	20	33	21	27	31

#### 4.4.2.1 Makespan and Number of AGVs

Applying the developed algorithms to testbed 1, results (best, mean, and worst fitness values) of each algorithm for 100 iterations are shown in Figure 4.2. Fitness values of all the algorithms range between 160 and 340. When comparing the algorithms' performance, HGP2 had a wider spread between its minimum and maximum fitness values, which it demonstrates the HGP2 capability of exploring a wider area in search of best results. PSO had a wide spread of results as well, but since its optimum answer is higher (worse) than the other EAs in this testbed, it is not comparable with HGP2. The smallest range of variation between minimum and maximum results was obtained using GA, which indicates its low exploration capability in this testbed.

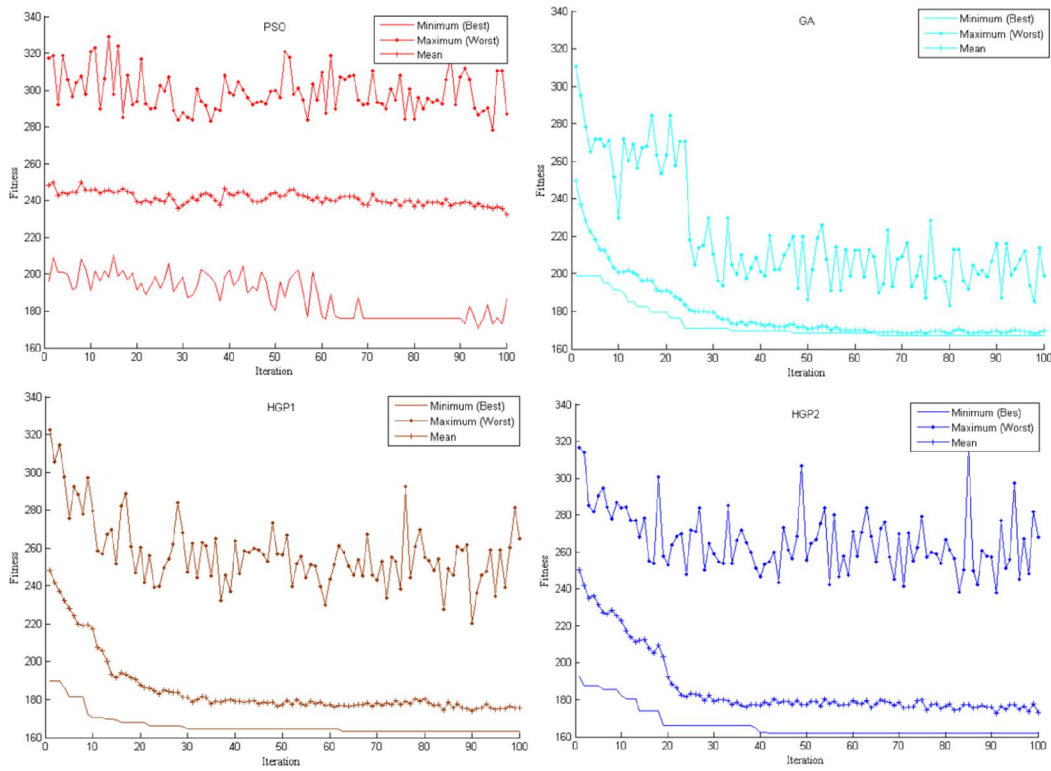


Figure 4.2: Performance of the four algorithms at testbed 1

To provide a comparison ground for the developed EAs, their best (minimum fitness value) performances are shown in Figure 4.3. The performance of all algorithms was satisfactory and all the four algorithms were proved successful in decreasing the makespan and the required number of AGVs. Of the four EAs, however, the optimized model using HGP2 converged at a faster rate and to a lower value. Solutions were seen to converge at about 40 iterations when using the HGP2, but in HGP1 it was about 63 iterations, GA about 75, and PSO about 95.



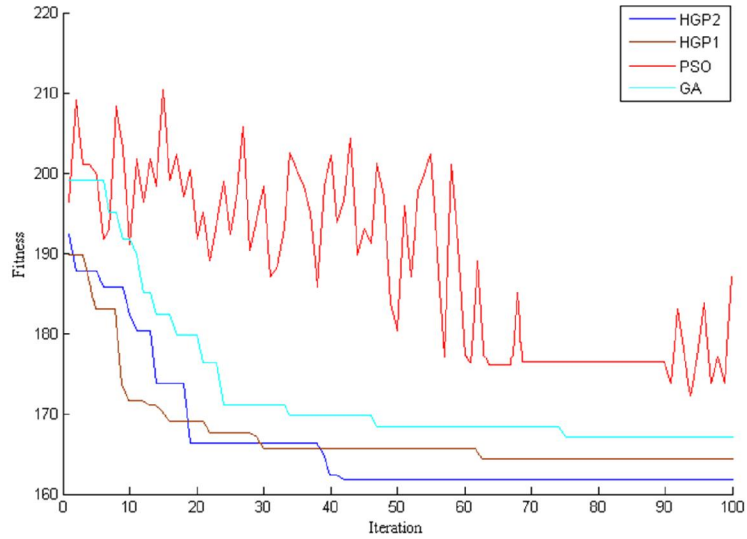


Figure 4.3: Best performance (minimum) of the four algorithms at testbed 1

The best, worst, and mean fitness values produced by each EA in this testbed are presented in Table 4.4. The makespan and number of AGVs corresponding with each fitness value are also reported. According to Table 4.4, the number of AGVs has been decreased in all the EAs. The smallest makespan was found in PSO with 4 AGVs, and the highest makespan in HGP1 using 2 AGVs. The obtained makespan values in GA and HGP2 varied between the highest and lowest makespan with 3 AGVs. Weighting the model objectives for attainment of one proper result out of all the fitness values obtained is highly pronounced in a situation as in this testbed. The fitness values in Table 4.4 have been calculated using equation 3.1. As  $MS \gg NA$ , the ratio used for this problem, based

on equation 3.2 would be  $\psi = \frac{\max(MS)}{\max(NA)}$ .  $\max(MS)$  was presumed to be equal to the

makespan when the sequence of operations under one AGV starts from the first operation of the first job to the last operation of the last job, which in testbed 1 it was equal to 844 minutes.  $\max(NA)$  was presumed equal to the whole number of operations which is 19

here.  $\delta_1$  is considered  $\frac{2}{3}$  for makespan because a higher weight is assigned to makespan,

but it can be changed subjectively to set equal weights to each of the objectives. The

fitness value for this testbed, based on equation 3.1, would be

$$f(x) = \frac{2}{3}(MS) + \frac{1}{3}\left(\frac{844}{19}\right)(NA).$$

As it is shown in Table 4.4, HGP2 obtained the best fitness value by 161.75, followed by HGP1 (164.94), GA (167.08), and PSO (172.56).

Standard deviation in the best results series and computational time of the algorithms are also demonstrated in Table 4.4. The smallest standard deviation was obtained from HGP2, followed by HGP1, GA, and PSO respectively. The obtained computational time of all the algorithms were nearly similar, but HGP1 had a slightly longer computational time and HGP2 had a shorter computational time compared with the other EAs applied.

Table 4.4: Test results of optimization algorithms at testbed 1 for hundred runs

Algorithm	Objectives	Best	Worst	Mean	Standard deviation	Computational time (second)
PSO	Fitness value	172.56	178.42	172.42	3.41	51.11
	Makespan (minute)	170	201	192	8.46	
	Number of AGVs	4	3	-	-	
GA	Fitness value	167.08	175.08	173.45	2.38	49.61
	Makespan (minute)	184	196	198	3.57	
	Number of AGVs	3	3	-	-	
HGP1	Fitness value	164.94	172.42	170.02	2.24	54.38
	Makespan (minute)	203	192	188.4	3.36	
	Number of AGVs	2	3	-	-	
HGP2	Fitness value	161.75	167.08	164.82	1.89	48.50
	Makespan (minute)	176	184	180.6	2.51	
	Number of AGVs	3	3	-	-	

To visualize differences between makespan and number of AGVs before and after the optimization by HGP2, a few random sequences have been tried before optimization and one of them has been chosen for presentation purpose at every testbed. Results have been graphically compared in Figures 4.4 and 4.5. Figure 4.4 shows the random sequence of operations before optimization which one AGV is assigned to each of the six jobs (O<sub>11</sub>, O<sub>21</sub>, O<sub>12</sub>, O<sub>31</sub>, O<sub>32</sub>, O<sub>22</sub>, O<sub>61</sub>, O<sub>62</sub>, O<sub>52</sub>, O<sub>41</sub>, O<sub>42</sub>, O<sub>33</sub>, O<sub>34</sub>, O<sub>13</sub>, O<sub>14</sub>, O<sub>23</sub>, O<sub>35</sub>, O<sub>43</sub>) obtaining the makespan of 365 minutes. Figure 4.5 demonstrates the optimized sequence of Figure 4.4, using only three AGVs with the makespan of 176 minutes obtained using HGP2, that

is changed to (O<sub>11</sub>, O<sub>31</sub>, O<sub>51</sub>, O<sub>21</sub>, O<sub>22</sub>, O<sub>32</sub>, O<sub>41</sub>, O<sub>33</sub>, O<sub>61</sub>, O<sub>13</sub>, O<sub>22</sub>, O<sub>34</sub>, O<sub>62</sub>, O<sub>14</sub>, O<sub>42</sub>, O<sub>35</sub>, O<sub>43</sub>, O<sub>23</sub>). In both the figures, every job and AGV is shown with a particular colour introduced below each figure. The dashed lines show the time that the part is waiting to be collected by the AGV or the time that AGV is waiting to be assigned to a task or being charged. Colour of the dashed lines are the same as the AGVs' colour.

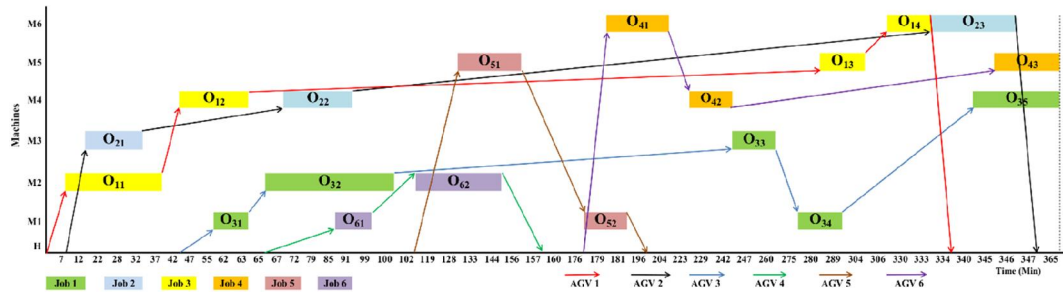


Figure 4.4: Operations' sequence before optimization – testbed 1

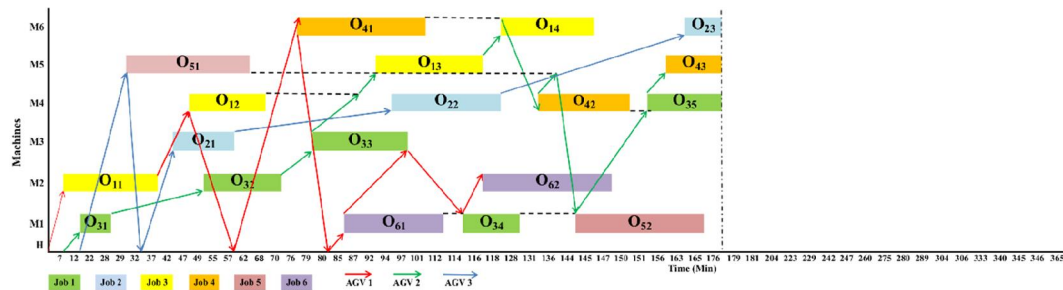


Figure 4.5: Operations' sequence after optimization by HGP2 – testbed 1

#### 4.4.2.2 AGVs' Battery Charge

After calculating the AGVs' battery charge consumption based on equations 3.17 to 3.20, the level of consumed battery charge of each AGV and corresponding battery charge utilization level both before and after optimization by all four algorithms are demonstrated in Figures 4.6 and 4.7, respectively. As it is shown in Figure 4.6, the overall battery charge consumption decreased dramatically, because the number of AGVs were decreased after the optimization—using any of the EAs. On the other hand, battery consumption of each AGV had slightly increased because the workload of omitted AGVs was undertaken by the remaining ones. However, it is worth mentioning that the increment in each AGV's battery consumption is notably low when compared with

overall reduction in battery consumption. Figures 4.6 also demonstrates the optimization effect in reducing the number of employed AGVs in the system, in which the AGVs reduction is compensated by improvements in battery utilization as shown in Figure 4.7.

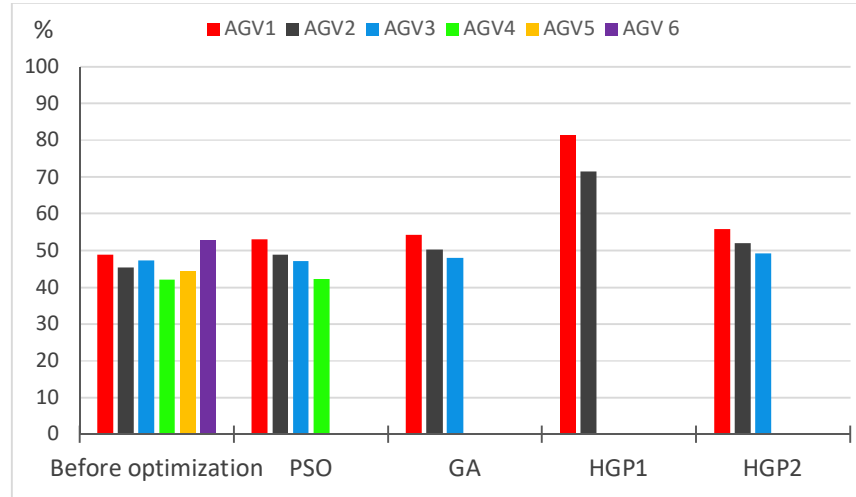


Figure 4.6: AGVs' battery charge consumption, before and after optimization

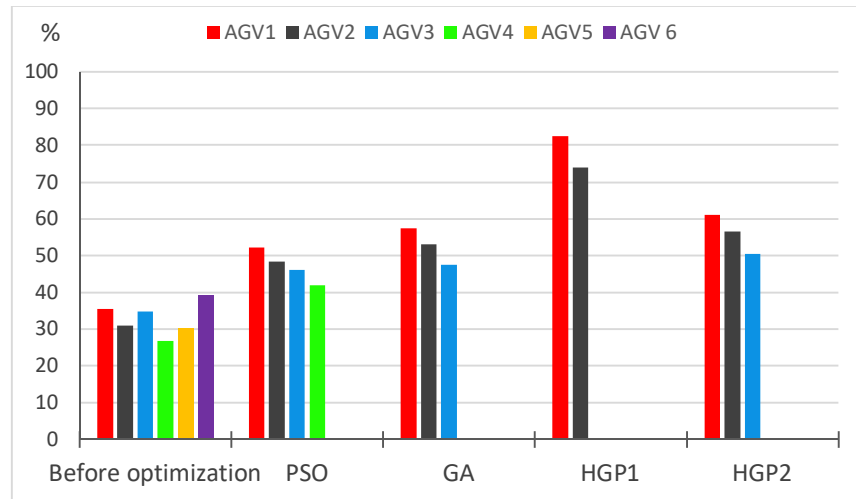


Figure 4.7: Battery charge utilization, before and after optimization – testbed 1

#### 4.4.2.3 AGVs' Specifications/Behavior

To investigate the optimization effect on AGVs' specifications/behaviour, the following specifications were explored. The studied specifications are AGVs' total running time (loaded and unloaded), idle time, and AGVs' operation efficiency computed using equations 3.21 to 3.27. In Figure 4.8, the AGVs' total running time prior to the

optimization is lower than that of after optimization shown in Figure 4.9, though the difference is not significant. The insignificant difference can be explained by the reduction in the number of AGVs after optimization, in which two or three AGVs undertake the workload of six AGVs. The idle time of AGVs after optimization decreased and led to a smaller makespan when compared with that of before optimization.

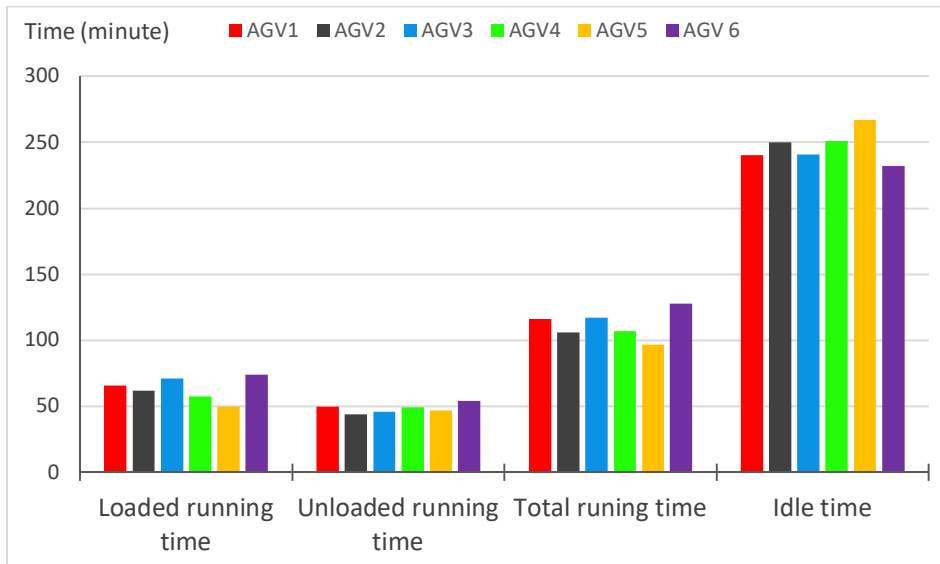


Figure 4.8: AGVs' specification before optimization

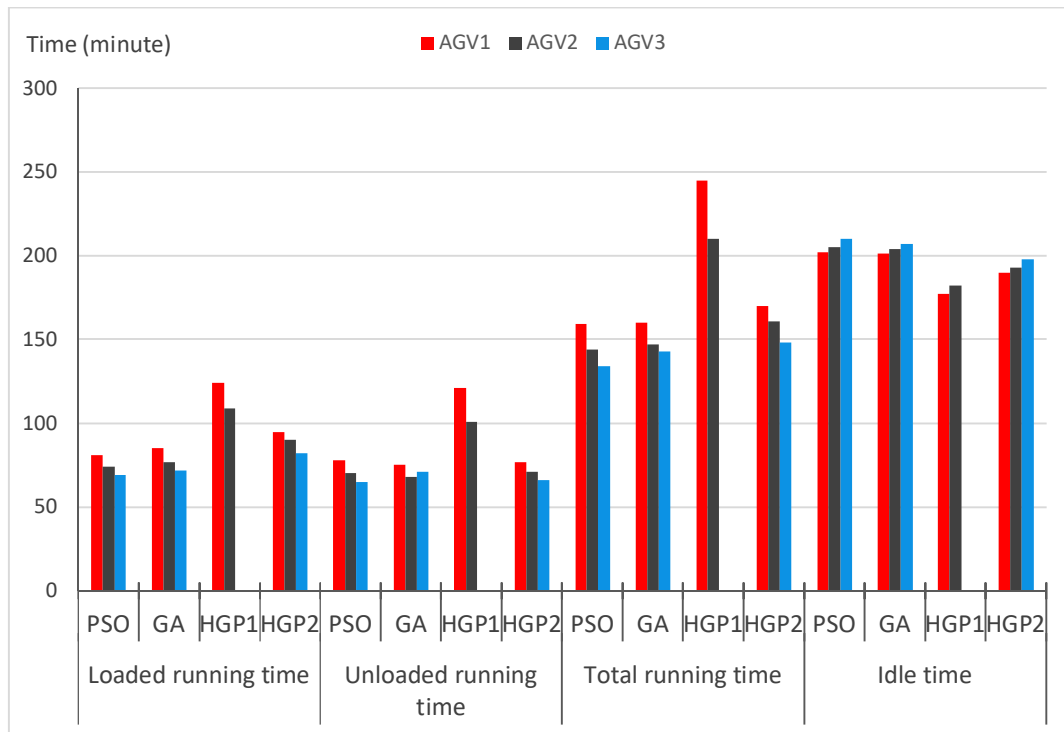


Figure 4.9: AGVs' specification after optimization

As it was shown earlier in Figure 4.6, the total level of consumed battery charge decreased after optimization; the fact that AGVs could do their work in a lesser time and with lesser battery charge consumption shows that AGVs' operation efficiency was consequently improved after the optimization. Figure 4.10 demonstrates the enhanced efficiency of AGVs' operation after the optimization compared with prior optimization status. AGVs' operation efficiency was considerably enhanced when compared with prior to optimization stage that all the efficiency measures were about 30% and below. After optimization, using all the EAs, all utilized AGVs' efficiency was escalated to above 40% limit, and many instances showed more than 50% of efficiency level. Before optimization, AGVs were deployed with no intention to use their highest potential and the AGV number one and number six showed the highest operation efficiency. After optimization, based on the scheduling model designed, tasks were sequentially appointed to AGVs based on the AGVs numerical order, so that AGV number one would have the highest operation efficiency level. This rule is to use the highest potential of the available/working AGVs to avoid addition of extra AGVs and expenses involved. Following this strategy, Figures 4.10 illustrates the highest and lowest efficiencies that were obtained respectively by the first and the last AGV under the application of any of the EAs.

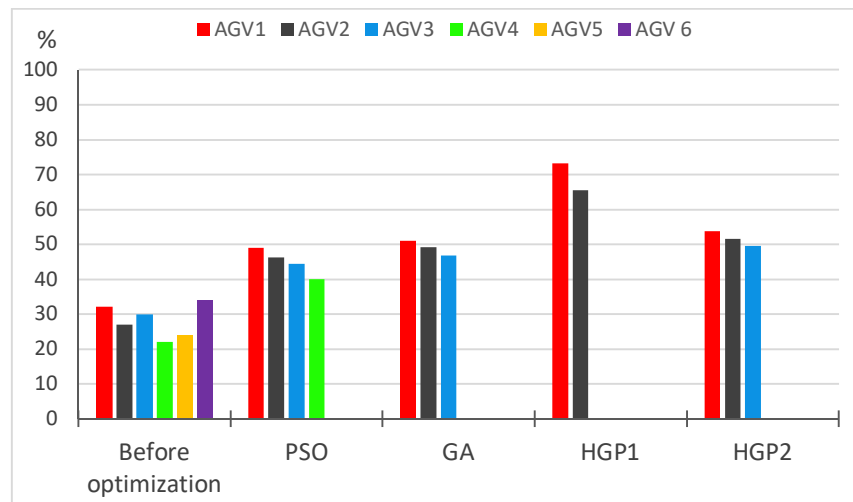


Figure 4.10: AGVs' operation efficiency before and after optimization

### 4.4.3 Performance at Testbed 2

This testbed was inclusive of 6 jobs ( $J_1, \dots, J_6$ ) processing on 12 machines ( $M_1, \dots, M_{12}$ ) and each job with 3 to 8 operations (overall 36 operations). This testbed had a longer distance among the machines compared with testbed 1. The Figure 4.11 explains the testbed's layout.

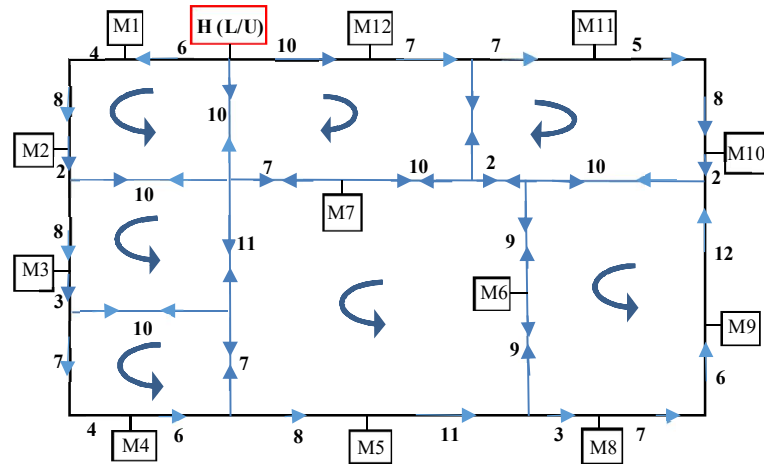


Figure 4.11: The layout of testbed 2

Table 4.5 shows the AGV travel time among L/U point and machines in this testbed, and Table 4.6 demonstrates the processing time of every operation on the machines. Inside routes are two-way and the surroundings are one-way routes.

Table 4.5: AGV travel time (minutes) between L/U points and machines

	L/U	M <sub>1</sub>	M <sub>2</sub>	M <sub>3</sub>	M <sub>4</sub>	M <sub>5</sub>	M <sub>6</sub>	M <sub>7</sub>	M <sub>8</sub>	M <sub>9</sub>	M <sub>10</sub>	M <sub>11</sub>	M <sub>12</sub>
L/U	0	6	18	28	42	36	38	17	50	63	37	24	10
M <sub>1</sub>	34	0	12	22	36	50	52	31	64	77	71	58	44
M <sub>2</sub>	22	28	0	10	24	38	40	19	52	65	59	46	32
M <sub>3</sub>	34	40	52	0	14	28	48	31	42	55	71	58	44
M <sub>4</sub>	34	40	52	42	0	14	34	31	28	41	71	58	44
M <sub>5</sub>	58	64	76	66	80	0	20	41	14	27	61	48	68
M <sub>6</sub>	38	46	58	46	60	54	0	21	12	25	41	28	48
M <sub>7</sub>	17	23	35	25	39	33	21	0	33	46	40	27	27
M <sub>8</sub>	64	70	82	72	86	80	44	47	0	13	67	54	74
M <sub>9</sub>	51	57	69	59	73	67	31	34	43	0	54	41	61
M <sub>10</sub>	41	47	59	49	63	57	21	24	33	46	0	31	51
M <sub>11</sub>	54	60	72	62	76	70	34	37	46	59	13	0	64
M <sub>12</sub>	44	50	62	52	66	60	28	27	40	53	27	14	0

Table 4.6: The processing time (minutes) of every operation on different machines

Job	1	1	1	1	1	1	1	1	2	2	2	2	2	2	3	3	3	3
Operation	1	2	3	4	5	6	7	8	1	2	3	4	5	6	1	2	3	4
Machine	M <sub>2</sub>	M <sub>6</sub>	M <sub>5</sub>	M <sub>8</sub>	M <sub>1</sub>	M <sub>12</sub>	M <sub>7</sub>	M <sub>5</sub>	M <sub>3</sub>	M <sub>4</sub>	M <sub>6</sub>	M <sub>11</sub>	M <sub>10</sub>	M <sub>9</sub>	M <sub>1</sub>	M <sub>2</sub>	M <sub>10</sub>	M <sub>4</sub>
Operation time	37	33	34	35	23	34	37	26	23	26	27	25	34	23	26	25	31	24

Job	3	3	3	3	4	4	4	4	4	4	5	5	5	6	6	6	6	6
Operation	5	6	7	8	1	2	3	4	5	6	1	2	3	1	2	3	4	5
Machine	M <sub>6</sub>	M <sub>7</sub>	M <sub>8</sub>	M <sub>11</sub>	M <sub>1</sub>	M <sub>7</sub>	M <sub>5</sub>	M <sub>12</sub>	M <sub>6</sub>	M <sub>8</sub>	M <sub>1</sub>	M <sub>7</sub>	M <sub>9</sub>	M <sub>3</sub>	M <sub>2</sub>	M <sub>10</sub>	M <sub>4</sub>	M <sub>11</sub>
Operation time	25	13	14	23	16	11	23	34	25	13	16	11	31	26	31	23	24	35

#### 4.4.3.1 Makespan and Number of AGVs

Performances of the four algorithms when applied to testbed 2 are shown in Figure 4.12. As in Figure 4.12, it was found that GA—in contrast with other EAs—had a steep incline in fitness measures of the first few iteration and then a smoother trend toward the best fitness value was seen. Other EAs applied to this testbed showed a nearly monotonic slope while reaching the best fit. The fitness values range from 1250 to 1850 for all the EAs in Figure 4.12.

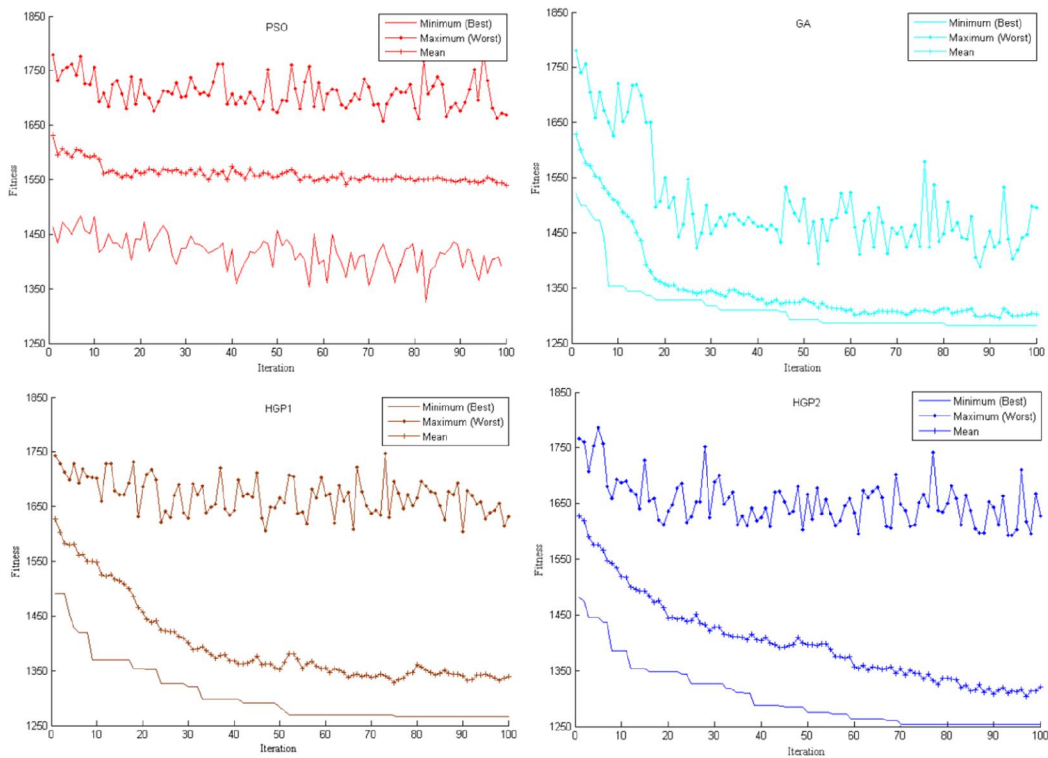


Figure 4.12: Performance of four algorithms at testbed 2

To compare the algorithms’ performance, their best results (minimum fitness value) are shown in Figure 4.13. All the four algorithms decreased the makespan and the required number of AGVs. However, the optimized model using HGP2 converged at a faster rate



and to a lower value. Solutions were seen to converge at about 70 generations when using HGP2, but in HGP1 it was about 75 generations, GA about 80, and PSO about 83.

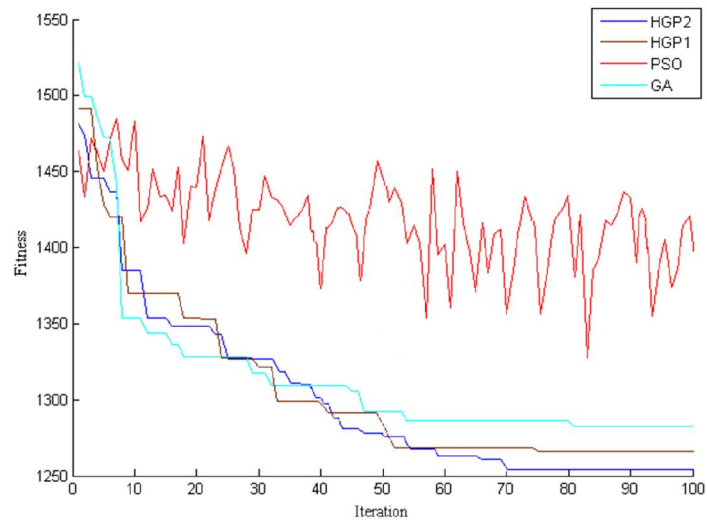


Figure 4.13: Best performance (minimum) of the four algorithms at testbed 2

The obtained results (best, worst, and mean fitness value) in this testbed are summarized in Table 4.7. The fitness function, based on equation 3.1, would be

$$f(x) = \frac{2}{3}(MS) + \frac{1}{3}\left(\frac{7829}{36}\right)(NA) \text{ for this testbed. According to the Table 4.7, the fitness}$$

value obtained by HGP2 (1253.96) shows the best fit, followed respectively by HGP1 (1272.22), GA (1282.63), and PSO (1327.29). In terms of the standard deviation in generation of the best (min) results series, HGP2 showed the smallest standard deviation followed by HGP1, GA, and PSO, respectively. The last column of the Table 4.7 shows the computational time of the algorithms. HGP1, in this regard, with approximately 10 seconds delay compared with the fastest EA had the longest computational time. Overall, HGP2 was the fastest EA and GA, PSO, and HGP1 were the next runner ups in line.

Table 4.7: Test results of optimization algorithms at testbed 2 for hundred runs

Algorithm	Objectives	Best	Worst	Mean	Standard deviation	Computational time (second)
PSO	Fitness value	1327.29	1411.29	1378.76	18.87	151.29
	Makespan (minute)	1556	1582	1633.20	28.31	
	Number of AGVs	4	5	-	-	
GA	Fitness value	1282.63	1407.787	1316.58	18.20	150.12
	Makespan (minute)	1489	1568	1539.93	22.81	
	Number of AGVs	4	5	-	-	
HGP1	Fitness value	1272.22	1334.22	1309.55	17.06	159.63
	Makespan (minute)	1474	1567	1530	27.09	
	Number of AGVs	4	4	-	-	
HGP2	Fitness value	1253.96	1336.63	1304.54	15.48	149.74
	Makespan (minute)	1446	1570	1521.86	26.22	
	Number of AGVs	4	4	-	-	

To elaborate the differences between before and after optimization by HGP2, Figure 4.14 shows the random sequence of operations before optimization (O<sub>11</sub>, O<sub>21</sub>, O<sub>31</sub>, O<sub>41</sub>, O<sub>51</sub>, O<sub>61</sub>, O<sub>12</sub>, O<sub>22</sub>, O<sub>32</sub>, O<sub>42</sub>, O<sub>52</sub>, O<sub>62</sub>, O<sub>13</sub>, O<sub>23</sub>, O<sub>33</sub>, O<sub>43</sub>, O<sub>53</sub>, O<sub>63</sub>, O<sub>14</sub>, O<sub>24</sub>, O<sub>34</sub>, O<sub>44</sub>, O<sub>64</sub>, O<sub>15</sub>, O<sub>25</sub>, O<sub>35</sub>, O<sub>45</sub>, O<sub>65</sub>, O<sub>16</sub>, O<sub>26</sub>, O<sub>36</sub>, O<sub>46</sub>, O<sub>17</sub>, O<sub>37</sub>, O<sub>18</sub>, O<sub>38</sub>) with the makespan of 1818 minutes using 6 AGVs. Figure 4.15 demonstrates the optimized sequence of Figure 4.14, using only 4 AGVs with the makespan of 1446 minutes obtained using HGP2, that is changed to (O<sub>11</sub>, O<sub>21</sub>, O<sub>41</sub>, O<sub>61</sub>, O<sub>22</sub>, O<sub>12</sub>, O<sub>23</sub>, O<sub>62</sub>, O<sub>24</sub>, O<sub>13</sub>, O<sub>63</sub>, O<sub>42</sub>, O<sub>64</sub>, O<sub>43</sub>, O<sub>14</sub>, O<sub>31</sub>, O<sub>25</sub>, O<sub>15</sub>, O<sub>32</sub>, O<sub>16</sub>, O<sub>33</sub>, O<sub>26</sub>, O<sub>17</sub>, O<sub>34</sub>, O<sub>51</sub>, O<sub>65</sub>, O<sub>18</sub>, O<sub>44</sub>, O<sub>35</sub>, O<sub>45</sub>, O<sub>36</sub>, O<sub>52</sub>, O<sub>37</sub>, O<sub>53</sub>, O<sub>46</sub>, O<sub>38</sub>).

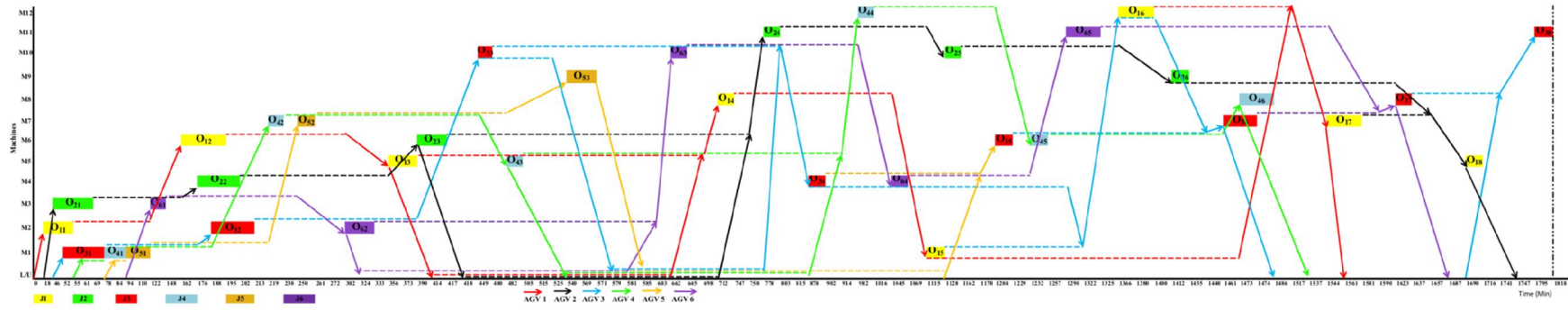


Figure 4.14: Operations' sequence before optimization – testbed 2

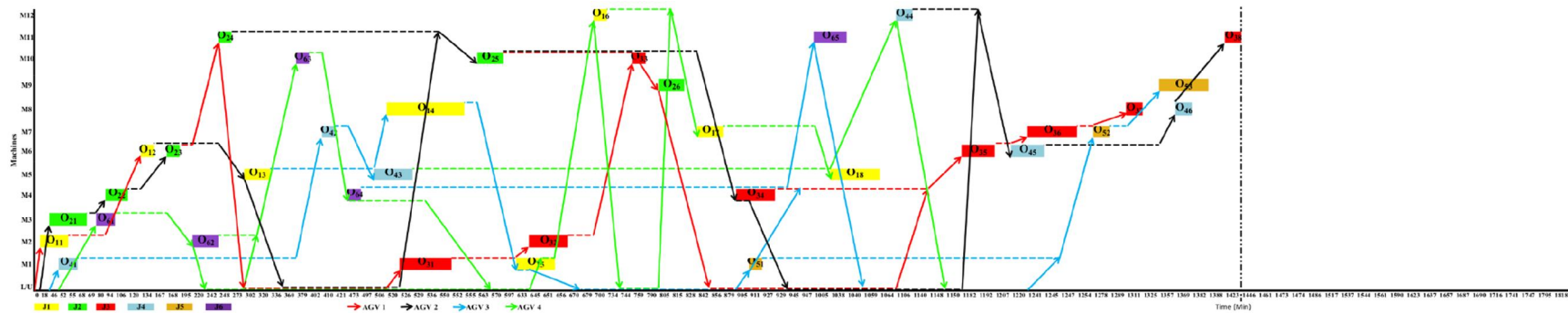


Figure 4.15: Operations' sequence after optimization by HGP2 – testbed 2

#### 4.4.3.2 AGVs' Battery Charge

The level of battery charge consumption of each AGV in testbed 2 are shown in Figure 4.16. Similar to the findings of testbed 1, the overall level of battery charge consumption of AGVs in the system was decreased by the reduction in number of utilized AGVs. Although there was a slight increase in the individual AGVs battery consumption after optimization, which relates to the overtaking of the tasks of omitted AGVs.

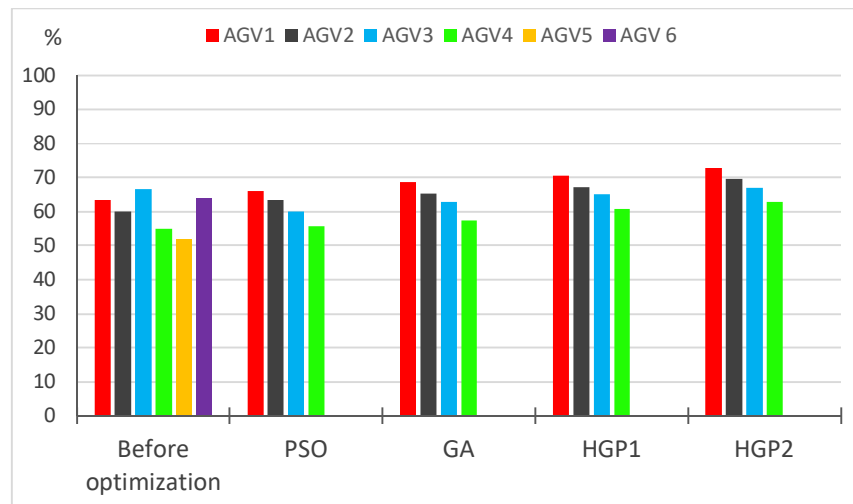


Figure 4.16: AGVs' battery charge consumption, before and after optimization

Figure 4.17 shows the status of battery charge utilization before and after optimization. Figure 4.17 clearly illustrates the improvements in AGVs battery charge utilization after the optimization. Every AGV was utilized to its highest potential and battery charge level after the optimization, so that the number of employed AGVs for the same volume of tasks reduced accordingly.

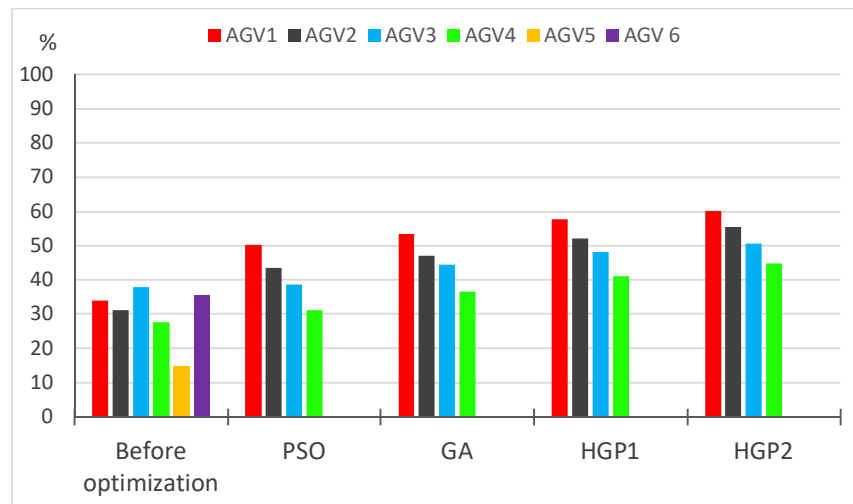


Figure 4.17: Battery charge utilization, before and after optimization – testbed 2

#### 4.4.3.3 AGVs' Specifications/Behavior

Figure 4.18 demonstrates the AGVs total running time prior to the optimization, which is lower than that of after optimization shown in Figure 4.19. Although the difference is not significant due to the reduction in the number of AGVs employed after optimization which the 4 AGVs undertake the workload of 6 previously used AGVs. As in Figure 4.19, the idle time of AGVs after optimization was also lessened dramatically and led to a smaller makespan when compared with that of before optimization shown in Figure 4.18.

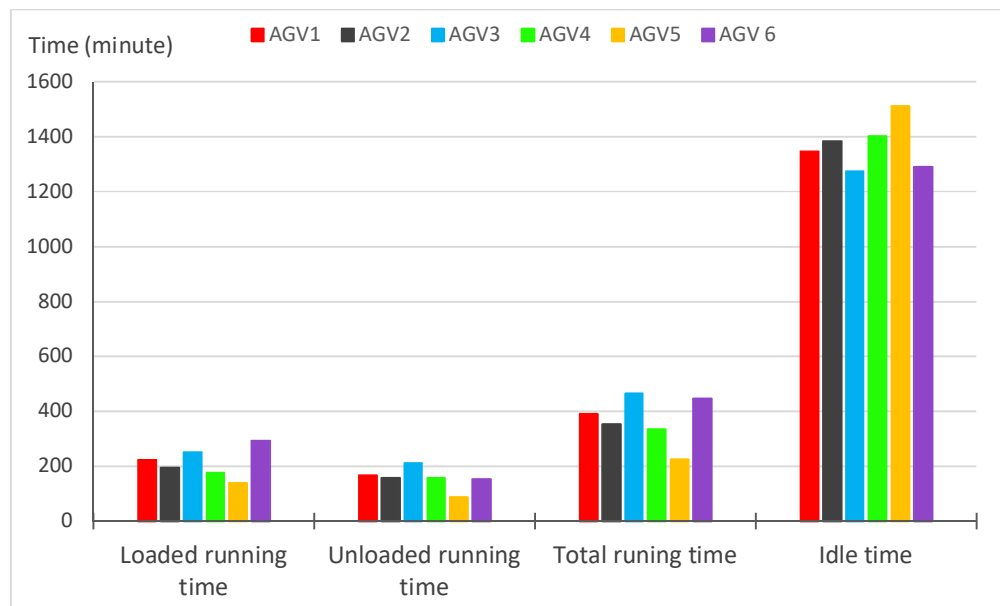


Figure 4.18: AGVs' specification before optimization

It was shown earlier in this testbed that after optimization the makespan and the number of AGVs were reduced; it is visualized here in Figure 4.19 that the AGVs' idle time have also been reduced.

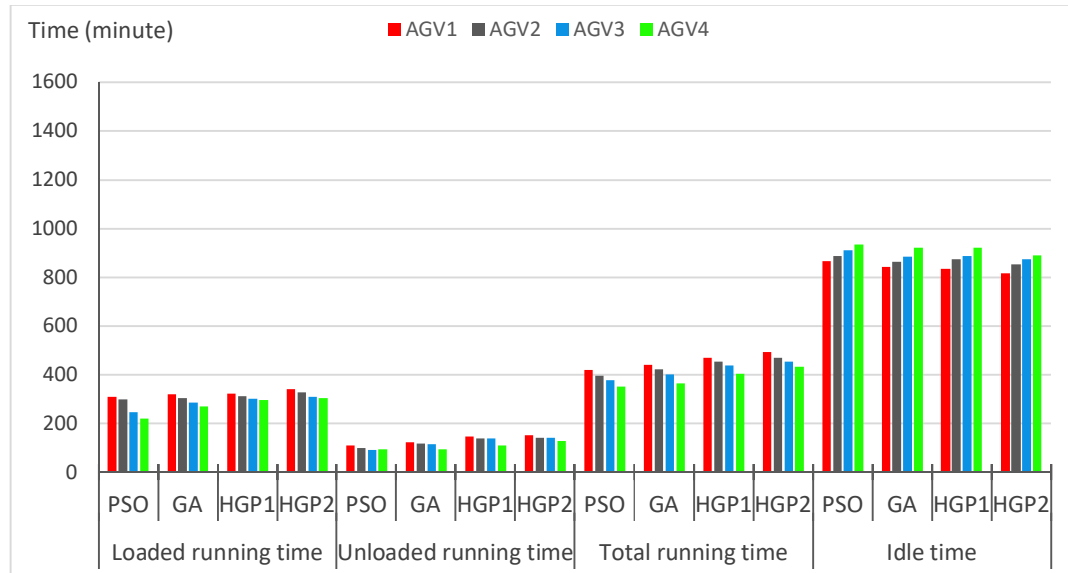


Figure 4.19: AGVs' specification after optimization

As it was shown earlier in Figure 4.16, the overall level of consumed battery charge decreased after optimization, so that AGVs' operation efficiency was consequently improved. Figure 4.20 demonstrates the enhanced efficiency level of AGVs' operation after optimization compared with prior optimization status. AGVs' operation efficiency was about 30% before optimization and it rose comparably after the optimization. After optimization, almost half the AGVs at all the EAs showed an operation efficiency level of more than 50%. The other half varied about 40% level of efficiency. Following the AGV utilization strategy as explained in testbed 1, Figures 4.20 illustrates the after-optimization efficiency level, in which the highest level was obtained by AGV1 and the lowest level by AGV4 in all the EAs. This finding is contrast with the AGV application strategy before optimization, where AGVs were added to the system without utilizing their highest potential so that many expensive AGVs would be employed.

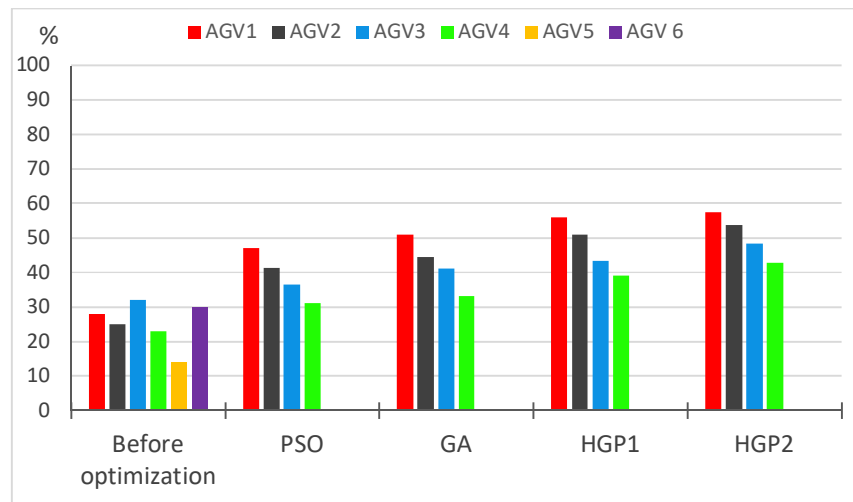


Figure 4.20: AGVs' operation efficiency before and after optimization

#### 4.4.4 Performance at Testbed 3

To create an even larger example for the model to be examined, the third example was designed with 15 jobs ( $J_1, \dots, J_{15}$ ) processing on 14 machines ( $M_1, \dots, M_{14}$ ), and each job with 3 to 5 operations (totally 60 operations). The testbed layout is illustrated in Figure 4.21 and AGV travel time among L/U point and machines are shown in Table 4.8. Table 4.9 demonstrates the processing time of every operation on the machines for testbed 3. In this testbed, the distances are shorter than the previous testbed. All the routes (either inside or surrounding) are two-way, which it shortens the distances among machines by equalizing the distances of going and returning to/from a machine. The processing time in this testbed is also smaller than the previous testbed.

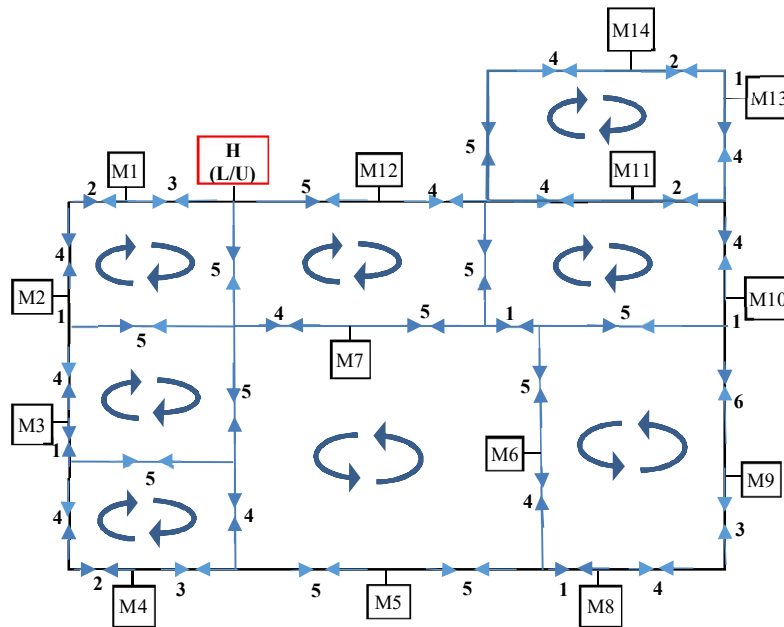


Figure 4.21: The layout of testbed 3

Table 4.8: AGV travel time (minutes) among L/U point and machines

Min	L/U	M <sub>1</sub>	M <sub>2</sub>	M <sub>3</sub>	M <sub>4</sub>	M <sub>5</sub>	M <sub>6</sub>	M <sub>7</sub>	M <sub>8</sub>	M <sub>9</sub>	M <sub>10</sub>	M <sub>11</sub>	M <sub>12</sub>	M <sub>13</sub>	M <sub>14</sub>
L/U	0	3	9	14	17	18	20	9	25	26	19	13	5	19	18
M <sub>1</sub>	3	0	6	11	18	21	23	12	28	29	22	16	8	21	21
M <sub>2</sub>	9	6	0	5	12	19	21	10	25	27	22	22	14	27	27
M <sub>3</sub>	14	11	5	0	7	14	23	13	20	27	25	27	19	30	32
M <sub>4</sub>	17	18	12	7	0	7	16	16	13	20	27	32	24	35	39
M <sub>5</sub>	18	21	19	14	7	0	9	17	6	13	20	24	23	28	29
M <sub>6</sub>	20	23	21	23	16	9	0	11	5	12	11	15	15	19	20
M <sub>7</sub>	9	12	10	13	16	17	11	0	16	17	12	14	14	20	19
M <sub>8</sub>	25	28	25	20	13	6	5	16	0	7	14	20	20	22	25
M <sub>9</sub>	26	29	27	27	20	13	12	17	7	0	7	13	21	15	18
M <sub>10</sub>	19	22	22	25	27	20	11	12	14	7	0	6	14	8	11
M <sub>11</sub>	13	16	22	27	32	24	15	14	20	13	6	0	8	6	9
M <sub>12</sub>	5	8	14	19	24	23	15	14	20	21	14	8	0	14	13
M <sub>13</sub>	19	21	27	30	35	28	19	20	22	15	8	6	14	0	3
M <sub>14</sub>	18	21	27	32	39	29	20	19	25	18	11	9	13	3	0

Table 4.9: The processing time (minutes) of every operation on the machines

Job	1	1	1	1	1	2	2	2	2	2	3	3	3	4	4	4
Operation	1	2	3	4	5	1	2	3	4	5	1	2	3	1	2	3
Machine	M <sub>7</sub>	M <sub>1</sub>	M <sub>9</sub>	M <sub>8</sub>	M <sub>3</sub>	M <sub>2</sub>	M <sub>10</sub>	M <sub>6</sub>	M <sub>14</sub>	M <sub>13</sub>	M <sub>5</sub>	M <sub>4</sub>	M <sub>12</sub>	M <sub>1</sub>	M <sub>3</sub>	M <sub>5</sub>
Operation time	19	21	14	10	11	15	21	22	19	23	30	26	13	6	12	19
Job	4	4	5	5	5	6	6	6	7	7	7	7	8	8	8	8
Operation	4	5	1	2	3	1	2	3	1	2	3	4	1	2	3	4
Machine	M <sub>7</sub>	M <sub>11</sub>	M <sub>10</sub>	M <sub>14</sub>	M <sub>12</sub>	M <sub>8</sub>	M <sub>6</sub>	M <sub>3</sub>	M <sub>2</sub>	M <sub>7</sub>	M <sub>12</sub>	M <sub>11</sub>	M <sub>9</sub>	M <sub>3</sub>	M <sub>4</sub>	M <sub>5</sub>
Operation time	14	18	29	26	1	12	24	17	29	16	20	18	9	21	24	10
Job	8	9	9	9	9	10	10	10	10	11	11	11	11	12	12	12
Operation	5	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3
Machine	M <sub>8</sub>	M <sub>10</sub>	M <sub>9</sub>	M <sub>5</sub>	M <sub>14</sub>	M <sub>5</sub>	M <sub>1</sub>	M <sub>8</sub>	M <sub>2</sub>	M <sub>7</sub>	M <sub>9</sub>	M <sub>8</sub>	M <sub>1</sub>	M <sub>6</sub>	M <sub>13</sub>	M <sub>11</sub>
Operation time	12	20	10	16	11	17	5	24	20	11	11	14	15	49	14	17
Job	13	13	13	13	14	14	14	14	14	14	15	15	15	15		
Operation	1	2	3	4	1	2	3	4	5	6	1	2	3	4		
Machine	M <sub>10</sub>	M <sub>1</sub>	M <sub>13</sub>	M <sub>12</sub>	M <sub>3</sub>	M <sub>4</sub>	M <sub>2</sub>	M <sub>10</sub>	M <sub>9</sub>	M <sub>12</sub>	M <sub>4</sub>	M <sub>8</sub>	M <sub>7</sub>	M <sub>11</sub>		
Operation time	29	30	8	14	12	11	9	10	10	19	15	21	4	14		



#### 4.4.4.1 Makespan and Number of AGVs

Following the same result presentation format, performances of the four algorithms at testbed 3 are shown in Figure 4.22. The fitness values obtained from the four algorithms vary between 820 and 1270. In this testbed, the range of PSO answers variation was shorter than that in testbed 1. HGP1 and HGP2 had nearly the same extent of exploration area in this testbed, and GA and PSO had a limited search area.

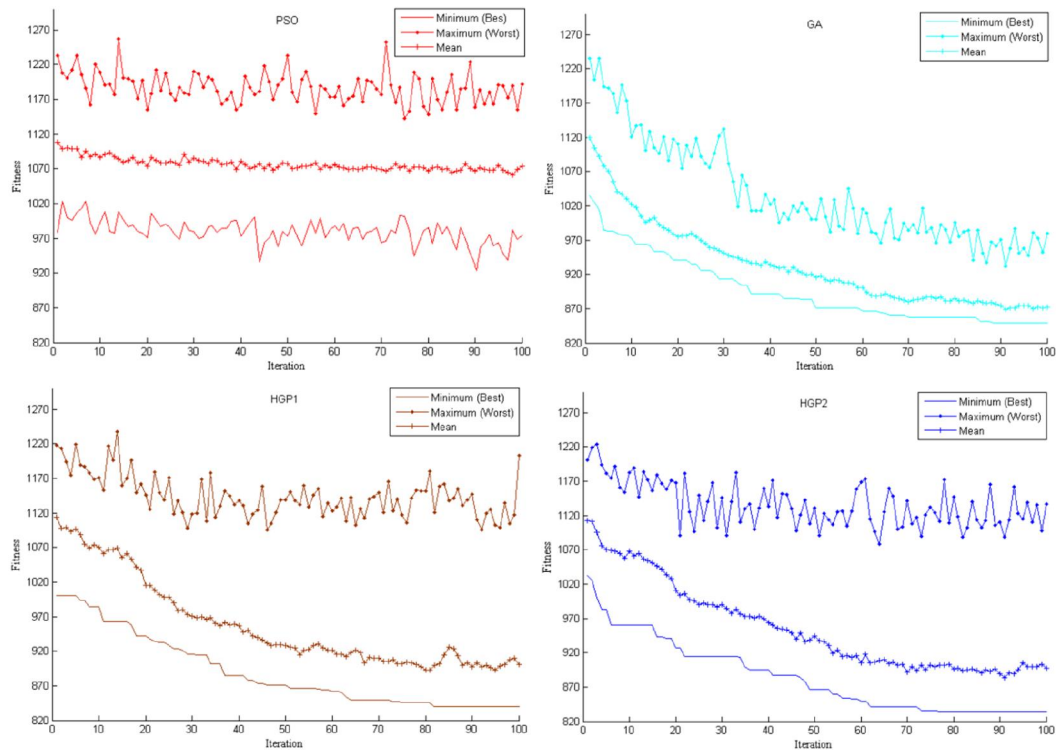


Figure 4.22: Performance of the four algorithms at testbed 3

For comparison purpose, series of the best (minimum fitness value) performances of developed EAs are shown in Figure 4.23. The performances of all algorithms were satisfactory and they were proved successful in decreasing the makespan and the required number of AGVs. Overall, the optimized model using HGP2 showed faster convergence in comparison with other EAs. Solutions were seen to reach the final solution after about 77 generations when using the HGP2, but in HGP1 it was about 83 generations, GA about 87, and PSO about 90.

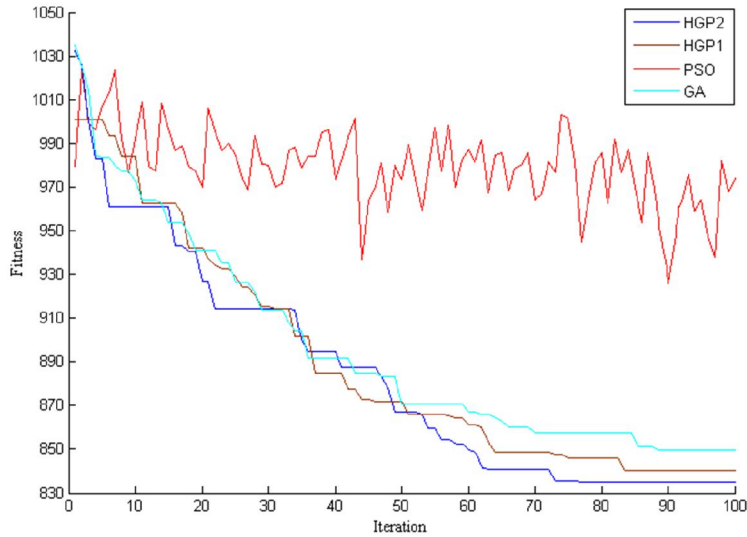


Figure 4.23: Best performance of the four algorithms at testbed 3

Out of all generations, the best, worst, and mean fitness values for the problem are reported in Table 4.10. The fitness function for this testbed, based on equation 3.1, would

be  $f(x) = \frac{2}{3}(MS) + \frac{1}{3}\left(\frac{4663}{62}\right)(NA)$ . As it is shown in Table 4.10, HGP2 produced the

best fitness value by 834.82, followed by HGP1 (840.15), then GA (849.48), and PSO (936.82).

Comparing the developed EAs in terms of the standard deviation in generating the best results (minimum fitness value) in 100 iterations, Table 4.10 shows that HGP2 has had the smallest standard deviation among the competing EAs. HGP1, GA, and PSO are the next runners up. The small standard deviation explains how the generated results scattered around the best results' mean. With this viewpoint in mind, HGP2 has proved to be more consistent in maintaining its performance throughout the iterations. With regard to the computational time of each EA in this testbed, it was found that HGP2 has been the fastest EA (Table 4.10). GA, PSO, and HGP1 were the next EAs with fastest computational time, respectively.

Table 4.10: Results of optimization algorithms at testbed 3 for hundred runs

Algorithm	Objectives	Best	Worst	Mean	Standard deviation	Computational time (second)
PSO	Fitness value	936.82	956.55	945.78	14.50	190.29
	Makespan (minute)	1142	1134	1140.4	21.17	
	Number of AGVs	7	8	-	-	
GA	Fitness value	849.48	891.48	871.16	12.07	189.74
	Makespan (minute)	1011	1074	1038.14	18.10	
	Number of AGVs	7	7	-	-	
HGP1	Fitness value	840.15	867.48	856.42	10.63	196.63
	Makespan (minute)	997	1038	1021.4	15.94	
	Number of AGVs	7	7	-	-	
HGP2	Fitness value	834.82	892.82	863.42	7.38	187.12
	Makespan (minute)	989	1076	1031.9	12.83	
	Number of AGVs	7	7	-	-	

To visualize differences between makespan and number of AGVs before and after the optimization, results have been graphically compared in Figures 4.24 and 4.25. Figure 4.24 shows the random sequence of operations before the optimization ( $O_{11}$ ,  $O_{21}$ ,  $O_{31}$ ,  $O_{41}$ ,  $O_{51}$ ,  $O_{61}$ ,  $O_{71}$ ,  $O_{81}$ ,  $O_{91}$ ,  $O_{10\_1}$ ,  $O_{11\_1}$ ,  $O_{12\_1}$ ,  $O_{13\_1}$ ,  $O_{14\_1}$ ,  $O_{15\_1}$ ,  $O_{12}$ ,  $O_{22}$ ,  $O_{32}$ ,  $O_{42}$ ,  $O_{52}$ ,  $O_{62}$ ,  $O_{72}$ ,  $O_{82}$ ,  $O_{92}$ ,  $O_{10\_2}$ ,  $O_{11\_2}$ ,  $O_{12\_2}$ ,  $O_{13\_2}$ ,  $O_{14\_2}$ ,  $O_{15\_2}$ ,  $O_{13}$ ,  $O_{23}$ ,  $O_{33}$ ,  $O_{43}$ ,  $O_{53}$ ,  $O_{63}$ ,  $O_{73}$ ,  $O_{83}$ ,  $O_{93}$ ,  $O_{10\_3}$ ,  $O_{11\_3}$ ,  $O_{12\_3}$ ,  $O_{13\_3}$ ,  $O_{14\_3}$ ,  $O_{15\_3}$ ,  $O_{14}$ ,  $O_{24}$ ,  $O_{44}$ ,  $O_{74}$ ,  $O_{84}$ ,  $O_{94}$ ,  $O_{10\_4}$ ,  $O_{11\_4}$ ,  $O_{13\_4}$ ,  $O_{14\_4}$ ,  $O_{15\_4}$ ,  $O_{15}$ ,  $O_{25}$ ,  $O_{45}$ ,  $O_{85}$ ,  $O_{14\_5}$ ,  $O_{14\_6}$ ) with the makespan of 1209 minutes and 12 AGVs. Figure 4.25 demonstrates the optimized sequence of Figure 4.24, using only 7 AGVs with the makespan of 989 minutes obtained using HGP2, that is changed to ( $O_{13\_1}$ ,  $O_{41}$ ,  $O_{14\_1}$ ,  $O_{81}$ ,  $O_{31}$ ,  $O_{42}$ ,  $O_{14\_2}$ ,  $O_{82}$ ,  $O_{61}$ ,  $O_{13\_2}$ ,  $O_{10\_1}$ ,  $O_{62}$ ,  $O_{15\_1}$ ,  $O_{11}$ ,  $O_{14\_3}$ ,  $O_{10\_2}$ ,  $O_{14\_4}$ ,  $O_{43}$ ,  $O_{63}$ ,  $O_{15\_2}$ ,  $O_{21}$ ,  $O_{51}$ ,  $O_{13\_3}$ ,  $O_{44}$ ,  $O_{15\_3}$ ,  $O_{22}$ ,  $O_{52}$ ,  $O_{12}$ ,  $O_{71}$ ,  $O_{32}$ ,  $O_{15\_4}$ ,  $O_{13}$ ,  $O_{91}$ ,  $O_{14}$ ,  $O_{14\_5}$ ,  $O_{23}$ ,  $O_{11\_1}$ ,  $O_{14\_6}$ ,  $O_{24}$ ,  $O_{72}$ ,  $O_{45}$ ,  $O_{15}$ ,  $O_{73}$ ,  $O_{83}$ ,  $O_{33}$ ,  $O_{10\_3}$ ,  $O_{84}$ ,  $O_{12\_1}$ ,  $O_{11\_2}$ ,  $O_{25}$ ,  $O_{12\_2}$ ,  $O_{74}$ ,  $O_{92}$ ,  $O_{11\_3}$ ,  $O_{10\_4}$ ,  $O_{93}$ ,  $O_{85}$ ,  $O_{11\_4}$ ,  $O_{12\_3}$ ,  $O_{94}$ ).

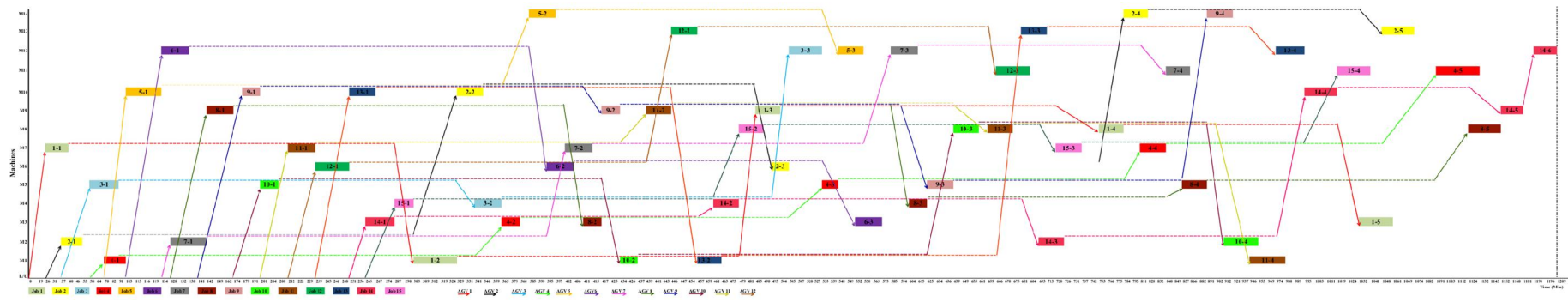


Figure 4.24: Operations' sequence before optimization – testbed 3

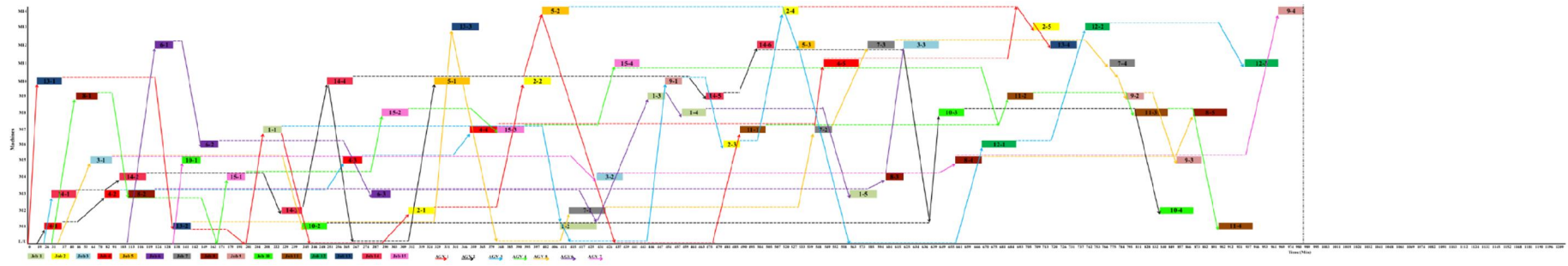


Figure 4.25: Operations' sequence after optimization by HGP2 – testbed 3

#### 4.4.4.2 AGVs' Battery Charge

The amount of battery charge consumption by each AGV both before and after the optimization using different algorithms are shown in Figure 4.26. The decrease in the overall battery charge consumption of AGVs in the system is evident from the notable reduction of AGVs number by each EA. However, similar to the previous testbeds, the amount of battery consumption by each AGV after optimization has slightly increased because they undertook the workload of omitted AGVs as well.

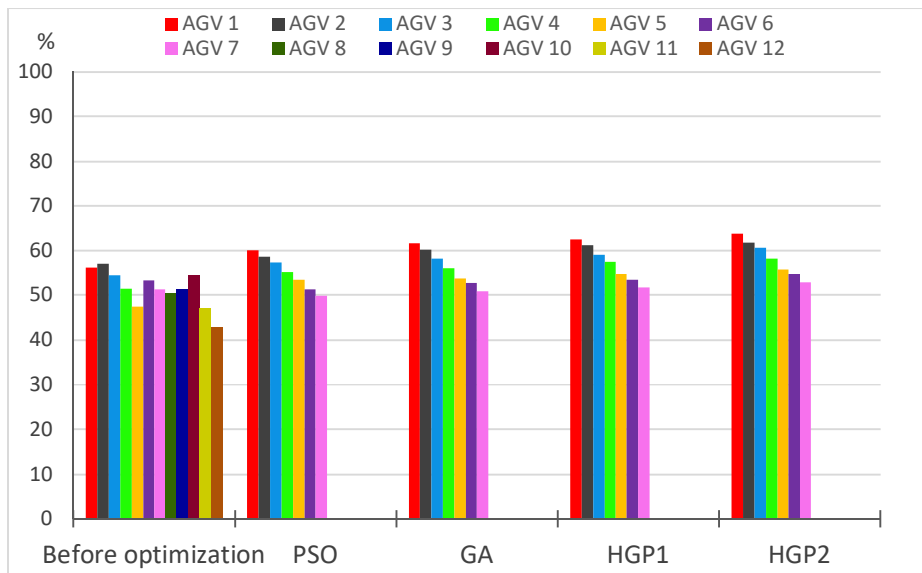


Figure 4.26: AGVs' battery charge consumption, before and after optimization

Figure 4.27 shows the percentage of AGV's battery charge utilization after optimization. It is found that the number of AGVs has been decreased after the optimization and the batteries of omitted AGVs were saved. Therefore, the battery charge utilization of all AGVs were heightened.

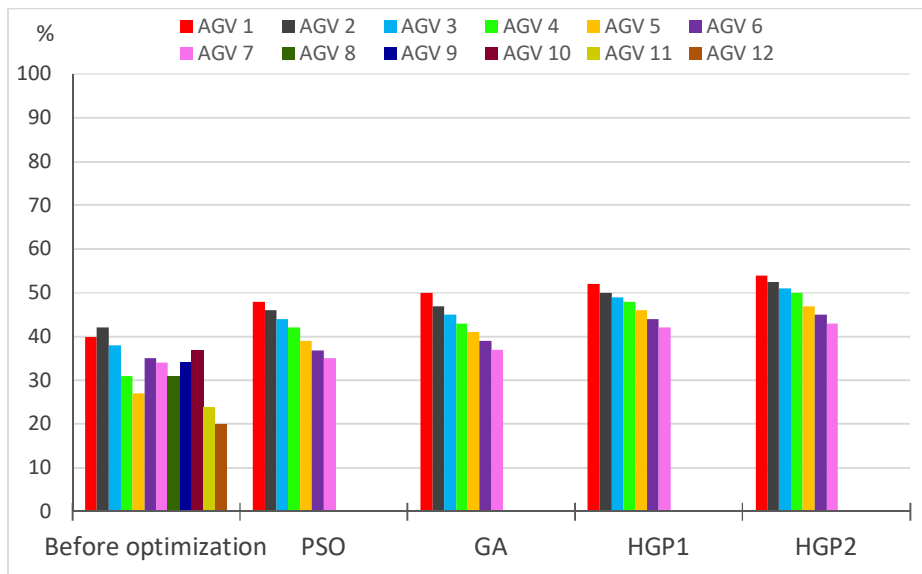


Figure 4.27: Battery charge utilization, before and after optimization – testbed 3

#### 4.4.4.3 AGVs' Specifications/Behavior

In Figure 4.28, the AGVs total running time prior to the optimization is lower than that of after optimization shown in Figure 4.29. Figure 4.29 explains the effect of optimization in reducing the number of AGVs utilized for performing the same volume of jobs. The total number of functioning AGVs has been reduced to 7 AGVs that undertook the workload of 12 AGVs performing prior to the optimization. It is again an indication of the AGVs utilization strategy of the model where AGV number 1 has the highest utilization measure. It also depicts the developed model strategy in assigning the AGVs to tasks, where AGVs are prioritized based on their numerical order. Figure 4.29 demonstrates the significant decrease in idle time of AGVs after the optimization, which has led to a smaller makespan when compared with that of before optimization.

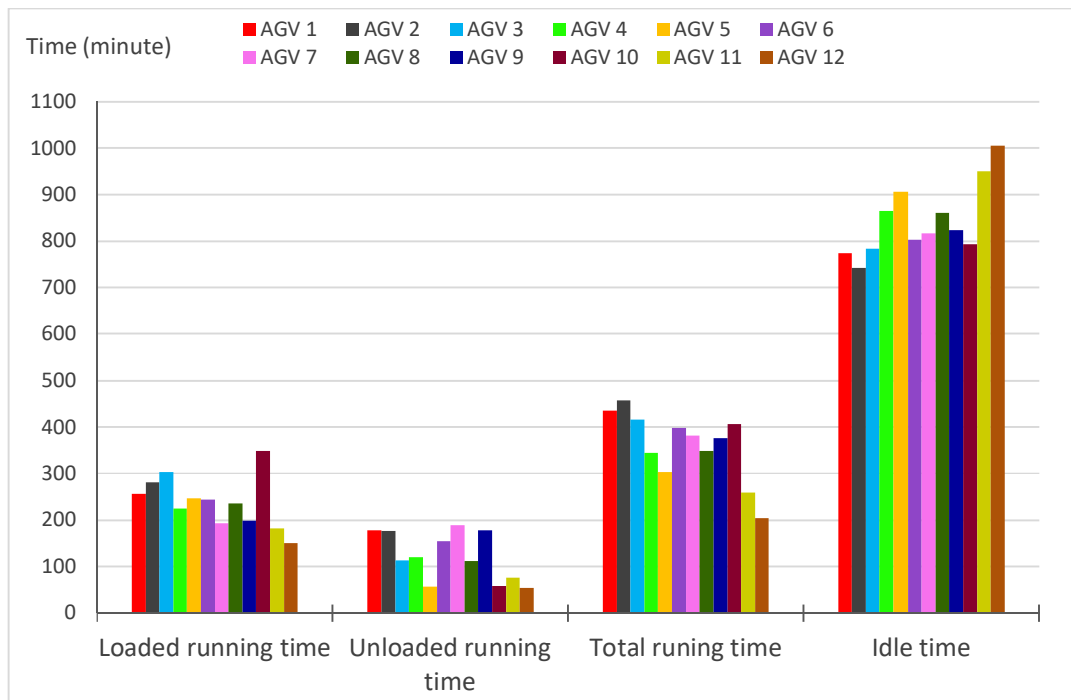


Figure 4.28: AGVs' specification before optimization

Overall, makespan, number of AGVs and their idle time have been reduced after the optimization and it is illustrated in Figure 4.29.

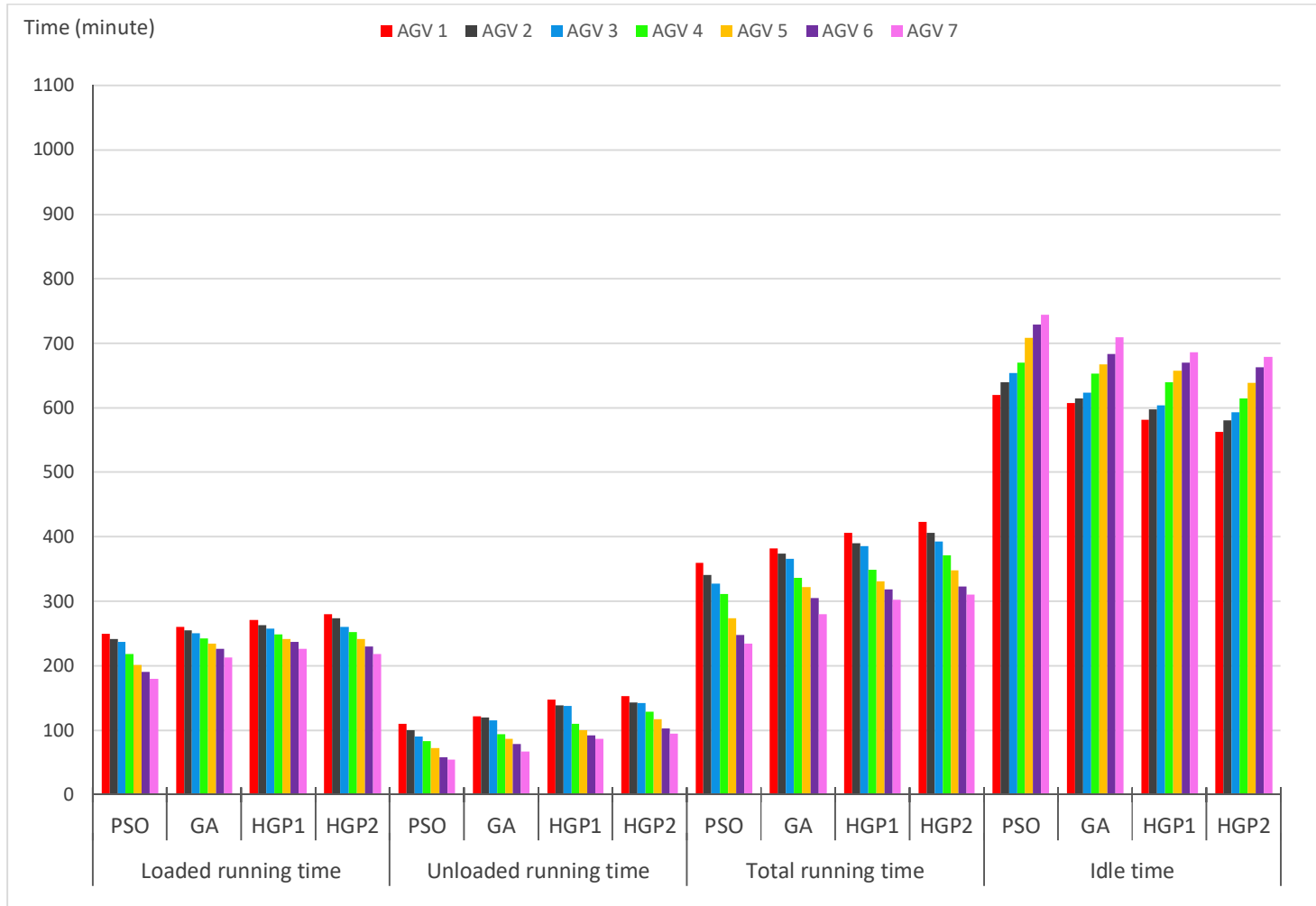


Figure 4.29: AGVs' specification after optimization



As it was shown earlier in Figure 4.26, the overall level of consumed battery charge decreased after the optimization, so that the AGVs' operation efficiency was consequently improved. Figure 4.30 demonstrates the enhanced efficiency of AGVs' operation after the optimization compared with prior optimization status. Analogous to other figures, response pattern of all the EAs to the optimization model is evident in Figure 4.30. It shows that the AGVs' operation efficiency before optimization is below 30% for all the AGVs and it is mounted above 40% for many of the AGVs after optimization, and above 30% for many others. AGVs were deployed with no intention to use their highest potential and the AGV number two then number one showed the highest operation efficiency before the optimization. However, following the optimization model strategy in assigning tasks to AGVs, the best potential of the first AGVs is used and addition of extra AGVs and expenses is avoided. Figure 4.30 clearly shows this systematic and cost-effective application of AGVs.

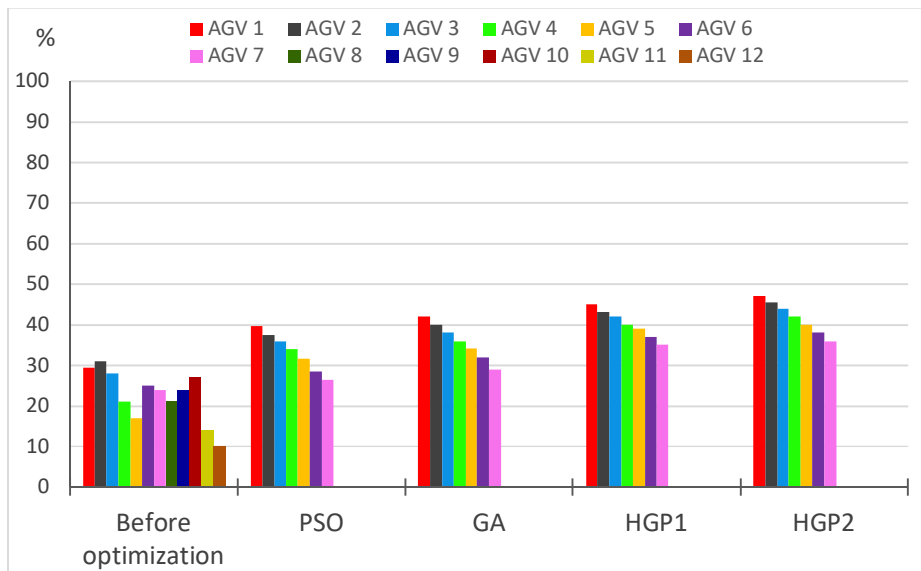


Figure 4.30: AGVs' operation efficiency before and after optimization

#### 4.4.5 Performance at Testbed 4

This numerical example was inclusive of 23 jobs ( $J_1, \dots, J_{23}$ ) processing on 17 machines ( $M_1, \dots, M_{17}$ ) and each job with 3 to 11 operations (totally 117 operations). Thus, this

example would represent a larger sized testbed with its layout being shown in Figure 4.31.

Table 4.11 shows the AGV travel time among L/U point and machines for testbed 4, and

Table 4.12 demonstrates the processing time of every operation on the machines. Similar

to testbed 3, all the routes in this testbed are also two-way and the distance of going and

returning to/from a machine are the same.

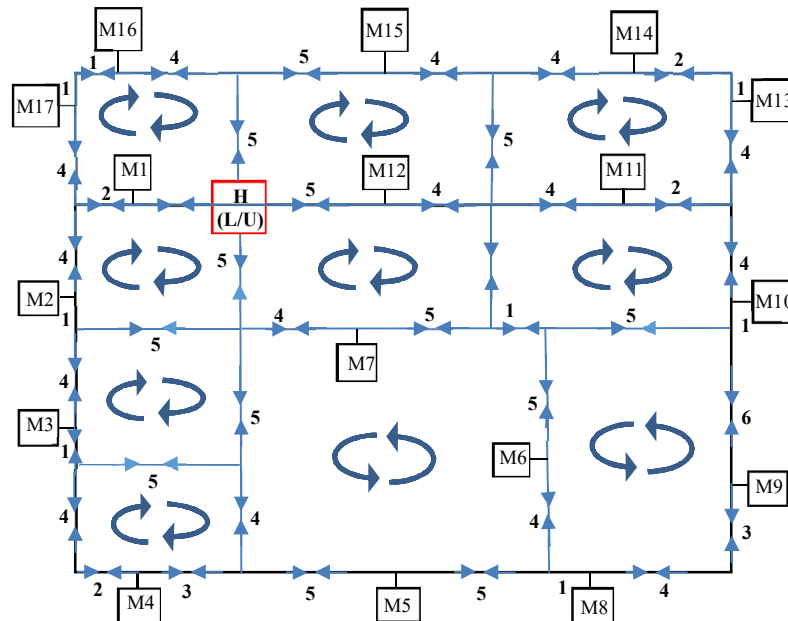


Figure 4.31: The layout of testbed 4

Table 4.11: AGV travel time (minutes) among L/U point and machines

Min	L/U	M <sub>1</sub>	M <sub>2</sub>	M <sub>3</sub>	M <sub>4</sub>	M <sub>5</sub>	M <sub>6</sub>	M <sub>7</sub>	M <sub>8</sub>	M <sub>9</sub>	M <sub>10</sub>	M <sub>11</sub>	M <sub>12</sub>	M <sub>13</sub>	M <sub>14</sub>	M <sub>15</sub>	M <sub>16</sub>	M <sub>17</sub>
L/U	0	3	9	14	17	18	20	9	25	26	19	13	5	19	18	10	9	9
M <sub>1</sub>	3	0	6	11	18	21	23	12	28	29	22	16	8	21	21	13	8	6
M <sub>2</sub>	9	6	0	5	12	19	21	10	25	27	22	22	14	27	27	19	10	8
M <sub>3</sub>	14	11	5	0	7	14	23	13	20	27	25	27	19	30	32	24	15	13
M <sub>4</sub>	17	18	12	7	0	7	16	16	13	20	27	32	24	35	39	27	22	20
M <sub>5</sub>	18	21	19	14	7	0	9	17	6	13	20	24	23	28	29	28	27	27
M <sub>6</sub>	20	23	21	23	16	9	0	11	5	12	11	15	15	19	20	20	29	29
M <sub>7</sub>	9	12	10	13	16	17	11	0	16	17	12	14	14	20	19	19	18	18
M <sub>8</sub>	25	28	25	20	13	6	5	16	0	7	14	20	20	22	25	25	34	34
M <sub>9</sub>	26	29	27	27	20	13	12	17	7	0	7	13	21	15	18	26	35	35
M <sub>10</sub>	19	22	22	25	27	20	11	12	14	7	0	6	14	8	11	19	28	28
M <sub>11</sub>	13	16	22	27	32	24	15	14	20	13	6	0	8	6	9	13	22	22
M <sub>12</sub>	5	8	14	19	24	23	15	14	20	21	14	8	0	14	13	13	14	14
M <sub>13</sub>	19	21	27	30	35	28	19	20	22	15	8	6	14	0	3	11	20	22
M <sub>14</sub>	18	21	27	32	39	29	20	19	25	18	11	9	13	3	0	8	17	19
M <sub>15</sub>	10	13	19	24	27	28	20	19	25	26	19	13	13	11	8	0	9	11
M <sub>16</sub>	9	8	10	15	22	27	29	18	34	35	28	22	14	20	17	9	0	2
M <sub>17</sub>	9	6	8	13	20	27	29	18	34	35	28	22	14	22	19	11	2	0

Table 4.12: The processing time (minutes) of every operation on the machines

Job	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	3	3	3	3	3
Operation	1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	1	2	3	4	5
Machine	M <sub>8</sub>	M <sub>3</sub>	M <sub>5</sub>	M <sub>17</sub>	M <sub>2</sub>	M <sub>12</sub>	M <sub>14</sub>	M <sub>1</sub>	M <sub>5</sub>	M <sub>17</sub>	M <sub>4</sub>	M <sub>16</sub>	M <sub>11</sub>	M <sub>10</sub>	M <sub>6</sub>	M <sub>11</sub>	M <sub>16</sub>	M <sub>9</sub>	M <sub>4</sub>	M <sub>3</sub>
Operation time	22	33	18	32	13	34	37	25	18	22	26	31	21	14	23	26	25	31	22	25
Job	3	3	3	3	3	3	4	4	4	4	4	4	5	5	5	6	6	6	6	6
Operation	6	7	8	9	10	11	1	2	3	4	5	6	1	2	3	1	2	3	4	5
Machine	M <sub>12</sub>	M <sub>7</sub>	M <sub>11</sub>	M <sub>14</sub>	M <sub>17</sub>	M <sub>10</sub>	M <sub>1</sub>	M <sub>16</sub>	M <sub>5</sub>	M <sub>12</sub>	M <sub>6</sub>	M <sub>8</sub>	M <sub>15</sub>	M <sub>7</sub>	M <sub>9</sub>	M <sub>13</sub>	M <sub>2</sub>	M <sub>10</sub>	M <sub>14</sub>	M <sub>11</sub>
Operation time	39	14	23	12	23	29	17	19	23	34	25	31	16	11	31	26	28	13	14	35
Job	7	7	7	7	8	8	8	9	9	9	9	9	10	10	10	11	11	11	11	11
Operation	1	2	3	4	1	2	3	1	2	3	4	5	1	2	3	1	2	3	4	5
Machine	M <sub>13</sub>	M <sub>15</sub>	M <sub>13</sub>	M <sub>9</sub>	M <sub>17</sub>	M <sub>1</sub>	M <sub>3</sub>	M <sub>8</sub>	M <sub>6</sub>	M <sub>13</sub>	M <sub>15</sub>	M <sub>9</sub>	M <sub>4</sub>	M <sub>8</sub>	M <sub>14</sub>	M <sub>12</sub>	M <sub>7</sub>	M <sub>16</sub>	M <sub>13</sub>	M <sub>15</sub>
Operation time	16	29	31	24	25	23	16	23	11	23	34	25	18	31	32	23	34	27	26	27
Job	12	12	12	13	13	13	13	13	14	14	14	14	14	15	15	15	15	15	15	
Operation	1	2	3	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	6	
Machine	M <sub>6</sub>	M <sub>4</sub>	M <sub>5</sub>	M <sub>14</sub>	M <sub>10</sub>	M <sub>9</sub>	M <sub>7</sub>	M <sub>8</sub>	M <sub>15</sub>	M <sub>16</sub>	M <sub>13</sub>	M <sub>1</sub>	M <sub>6</sub>	M <sub>3</sub>	M <sub>4</sub>	M <sub>8</sub>	M <sub>11</sub>	M <sub>10</sub>	M <sub>9</sub>	
Operation time	26	14	25	28	23	16	25	31	24	25	23	14	23	16	11	23	34	18	23	
Job	16	16	16	17	17	17	17	18	18	18	18	18	19	19	19	20	20	20	20	20
Operation	1	2	3	1	2	3	4	1	2	3	4	5	1	2	3	1	2	3	4	
Machine	M <sub>5</sub>	M <sub>11</sub>	M <sub>14</sub>	M <sub>7</sub>	M <sub>17</sub>	M <sub>2</sub>	M <sub>10</sub>	M <sub>11</sub>	M <sub>16</sub>	M <sub>17</sub>	M <sub>15</sub>	M <sub>3</sub>	M <sub>8</sub>	M <sub>9</sub>	M <sub>16</sub>	M <sub>13</sub>	M <sub>2</sub>	M <sub>12</sub>	M <sub>10</sub>	
Operation time	16	12	31	24	30	23	24	15	16	11	23	33	25	13	16	31	23	12	13	
Job	20	20	20	20	20	20	21	21	21	21	21	22	22	22	22	23	23	23		
Operation	5	6	7	8	9	10	1	2	3	4	5	1	2	3	4	1	2	3		
Machine	M <sub>16</sub>	M <sub>11</sub>	M <sub>9</sub>	M <sub>14</sub>	M <sub>13</sub>	M <sub>6</sub>	M <sub>7</sub>	M <sub>1</sub>	M <sub>4</sub>	M <sub>12</sub>	M <sub>3</sub>	M <sub>17</sub>	M <sub>10</sub>	M <sub>5</sub>	M <sub>13</sub>	M <sub>14</sub>	M <sub>9</sub>	M <sub>2</sub>		
Operation time	26	28	11	18	21	23	17	29	9	18	12	19	33	30	15	27	24	14		

#### 4.4.5.1 Makespan and Number of AGVs

Performance of the four algorithms at testbed 4 is shown in Figure 4.32. The amplitude of fitness values variation is from 1630 to 2280. Similar to testbed 2 and 3, PSO and GA algorithms had a smaller exploration area compared to the hybrids.

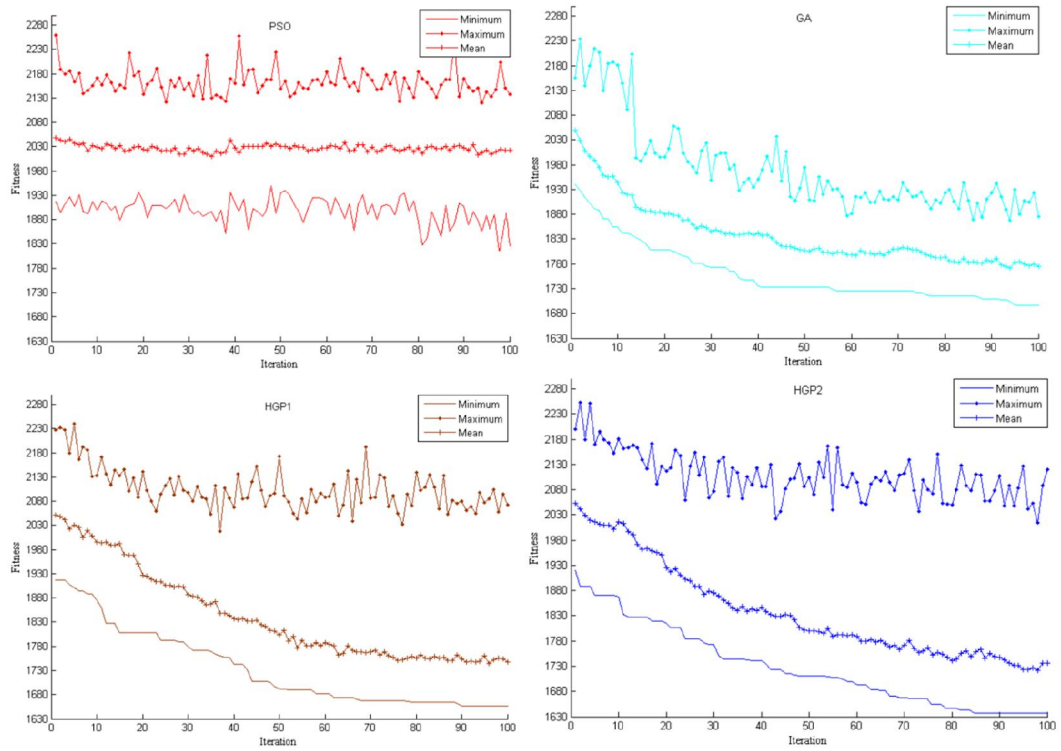


Figure 4.32: Performance of four algorithms at testbed 4

The best/minimum fitness values of the four algorithms at testbed 4 are shown in Figure 4.33. All the algorithms were successful in decreasing the makespan and the required number of AGVs. The optimized model using HGP2 converged at a faster rate and to a lower value compared with other EAs. Solutions convergence happened after about 85 generations in HGP2, 90 generations in HGP1, 95 in GA, and 97 in PSO.

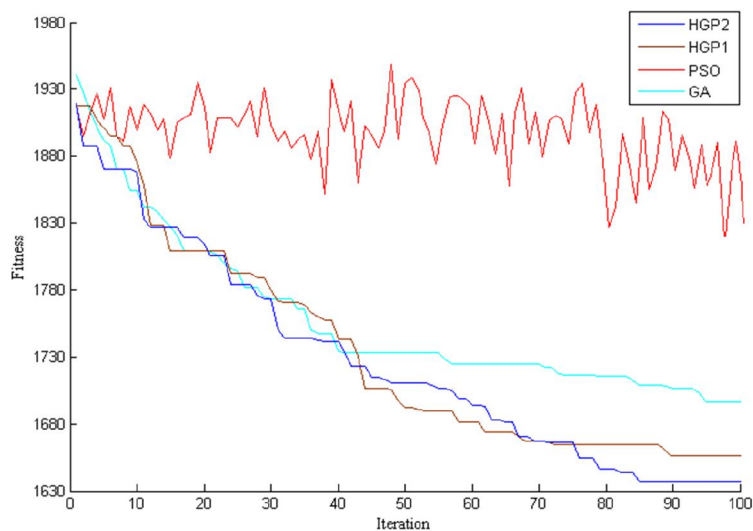


Figure 4.33: Best performance (minimum) of the four algorithms at testbed 4

Out of all generations above, the best, worst, and the mean results achieved from the overall fitness function for testbed 4 are tabulated in Table 4.13. The makespan, and number of AGVs for each result is shown as well. The fitness value for this testbed, based on equation 3.1, would be  $f(x) = \frac{2}{3}(MS) + \frac{1}{3}\left(\frac{8086}{116}\right)(NA)$ . As it is shown in Table 4.13, HGP2 has attained the best fitness value by 1542.83, followed by HGP1 (1555.50), then GA (1573.50), and PSO (1635.33). With regard to the results given in Table 4.13, it is evident that HGP2 has obtained the best (minimum fitness value) result, even in terms of the standard deviation. The last column of the Table 4.13 shows the computational time of the algorithms that highlights the HGP2 as the fastest EA applied. GA, PSO and HGP1 were the runners up to the HGP2 computational time in this testbed.

Table 4.13: Test results of optimization algorithms at testbed 4 for hundred runs

Algorithm	Objectives	Best	Worst	Mean	Standard deviation	Computational time (second)
PSO	Fitness value	1635.33	1750.66	1686.02	29.52	391.29
	Makespan (minute)	2000	2173	2105.8	39.93	
	Number of AGVs	13	13	-	-	
GA	Fitness value	1573.50	1722.83	1666.82	26.62	389.32
	Makespan (minute)	1942	2166	2082	27.43	
	Number of AGVs	12	12	-	-	
HGP1	Fitness value	1555.50	1694.16	1618.83	17.31	401.42
	Makespan (minute)	1915	2123	2010	25.97	
	Number of AGVs	12	12	-	-	
HGP2	Fitness value	1542.83	1637.49	1603.49	14.53	387.65
	Makespan (minute)	1896	2038	1987	21.29	
	Number of AGVs	12	12	-	-	

To demonstrate the effect of optimization on the makespan and number of AGVs applied in the testbed, Figures 4.34 and 4.35 respectively show the before and after optimization status. Figure 4.34 shows the random sequence of operations before optimization ( $O_{11}$ ,  $O_{21}$ ,  $O_{31}$ ,  $O_{41}$ ,  $O_{51}$ ,  $O_{61}$ ,  $O_{71}$ ,  $O_{81}$ ,  $O_{91}$ ,  $O_{10\_1}$ ,  $O_{11\_1}$ ,  $O_{12\_1}$ ,  $O_{13\_1}$ ,  $O_{14\_1}$ ,  $O_{15\_1}$ ,  $O_{16\_1}$ ,  $O_{17\_1}$ ,  $O_{18\_1}$ ,  $O_{19\_1}$ ,  $O_{20\_1}$ ,  $O_{21\_1}$ ,  $O_{22\_1}$ ,  $O_{23\_1}$ ,  $O_{12}$ ,  $O_{22}$ ,  $O_{32}$ ,  $O_{42}$ ,  $O_{52}$ ,  $O_{62}$ ,  $O_{72}$ ,  $O_{82}$ ,  $O_{92}$ ,  $O_{10\_2}$ ,  $O_{11\_2}$ ,  $O_{12\_2}$ ,  $O_{13\_2}$ ,  $O_{14\_2}$ ,  $O_{15\_2}$ ,  $O_{16\_2}$ ,  $O_{17\_2}$ ,  $O_{18\_2}$ ,  $O_{19\_2}$ ,  $O_{20\_2}$ ,  $O_{21\_2}$ ,  $O_{22\_2}$ ,  $O_{23\_2}$ ,  $O_{13}$ ,

O<sub>23</sub>, O<sub>33</sub>, O<sub>43</sub>, O<sub>53</sub>, O<sub>63</sub>, O<sub>73</sub>, O<sub>83</sub>, O<sub>93</sub>, O<sub>10\_3</sub>, O<sub>11\_3</sub>, O<sub>12\_3</sub>, O<sub>13\_3</sub>, O<sub>14\_3</sub>, O<sub>15\_3</sub>, O<sub>16\_3</sub>, O<sub>17\_3</sub>, O<sub>18\_3</sub>, O<sub>19\_3</sub>, O<sub>20\_3</sub>, O<sub>21\_3</sub>, O<sub>22\_3</sub>, O<sub>23\_3</sub>, O<sub>14</sub>, O<sub>24</sub>, O<sub>34</sub>, O<sub>44</sub>, O<sub>64</sub>, O<sub>74</sub>, O<sub>94</sub>, O<sub>11\_4</sub>, O<sub>13\_4</sub>, O<sub>14\_4</sub>, O<sub>15\_4</sub>, O<sub>17\_4</sub>, O<sub>18\_4</sub>, O<sub>19\_4</sub>, O<sub>20\_4</sub>, O<sub>22\_4</sub>, O<sub>15</sub>, O<sub>25</sub>, O<sub>35</sub>, O<sub>45</sub>, O<sub>65</sub>, O<sub>95</sub>, O<sub>11\_5</sub>, O<sub>13\_5</sub>, O<sub>14\_5</sub>, O<sub>15\_5</sub>, O<sub>18\_5</sub>, O<sub>20\_5</sub>, O<sub>21\_5</sub>, O<sub>16</sub>, O<sub>26</sub>, O<sub>36</sub>, O<sub>46</sub>, O<sub>15\_6</sub>, O<sub>20\_6</sub>, O<sub>17</sub>, O<sub>37</sub>, O<sub>20\_7</sub>, O<sub>1\_8</sub>, O<sub>3\_8</sub>, O<sub>20\_8</sub>, O<sub>1\_9</sub>, O<sub>3\_9</sub>, O<sub>20\_9</sub>, O<sub>3\_10</sub>, O<sub>20\_10</sub>, O<sub>3\_11</sub>) with the makespan of 2615 minutes and 23 AGVs.

Figure 4.35 demonstrates the optimized sequence of Figure 4.34 using HGP2 which has been the best performing EA among the applied EAs and used here as an example. HGP2 has optimized the sequence in Figure 4.34 using only 12 AGVs with the makespan of 1896 minutes obtained, and the sequence is changed to (O<sub>61</sub>, O<sub>31</sub>, O<sub>17\_1</sub>, O<sub>14\_1</sub>, O<sub>81</sub>, O<sub>11</sub>, O<sub>32</sub>, O<sub>62</sub>, O<sub>41</sub>, O<sub>12</sub>, O<sub>33</sub>, O<sub>17\_1</sub>, O<sub>42</sub>, O<sub>18\_1</sub>, O<sub>21</sub>, O<sub>15\_1</sub>, O<sub>34</sub>, O<sub>11\_1</sub>, O<sub>21\_1</sub>, O<sub>43</sub>, O<sub>18\_2</sub>, O<sub>44</sub>, O<sub>15\_2</sub>, O<sub>35</sub>, O<sub>20\_1</sub>, O<sub>16\_1</sub>, O<sub>21\_2</sub>, O<sub>13</sub>, O<sub>16\_2</sub>, O<sub>36</sub>, O<sub>91</sub>, O<sub>13\_1</sub>, O<sub>19\_1</sub>, O<sub>14\_2</sub>, O<sub>22</sub>, O<sub>14</sub>, O<sub>71</sub>, O<sub>10\_1</sub>, O<sub>15\_3</sub>, O<sub>19\_2</sub>, O<sub>15</sub>, O<sub>20\_2</sub>, O<sub>14\_3</sub>, O<sub>20\_3</sub>, O<sub>16</sub>, O<sub>51</sub>, O<sub>23</sub>, O<sub>20\_4</sub>, O<sub>52</sub>, O<sub>15\_4</sub>, O<sub>92</sub>, O<sub>23\_1</sub>, O<sub>11\_2</sub>, O<sub>21\_3</sub>, O<sub>18\_3</sub>, O<sub>15\_5</sub>, O<sub>12\_1</sub>, O<sub>24</sub>, O<sub>72</sub>, O<sub>16\_3</sub>, O<sub>63</sub>, O<sub>11\_3</sub>, O<sub>10\_2</sub>, O<sub>15\_6</sub>, O<sub>45</sub>, O<sub>19\_3</sub>, O<sub>13\_2</sub>, O<sub>64</sub>, O<sub>93</sub>, O<sub>14\_4</sub>, O<sub>37</sub>, O<sub>20\_5</sub>, O<sub>12\_2</sub>, O<sub>23\_2</sub>, O<sub>94</sub>, O<sub>46</sub>, O<sub>38</sub>, O<sub>20\_6</sub>, O<sub>10\_3</sub>, O<sub>53</sub>, O<sub>21\_4</sub>, O<sub>39</sub>, O<sub>22\_1</sub>, O<sub>65</sub>, O<sub>20\_7</sub>, O<sub>21\_5</sub>, O<sub>20\_8</sub>, O<sub>22\_2</sub>, O<sub>13\_3</sub>, O<sub>18\_4</sub>, O<sub>13\_4</sub>, O<sub>14\_5</sub>, O<sub>17</sub>, O<sub>3\_10</sub>, O<sub>18\_5</sub>, O<sub>11\_4</sub>, O<sub>3\_11</sub>, O<sub>22\_3</sub>, O<sub>25</sub>, O<sub>20\_9</sub>, O<sub>82</sub>, O<sub>13\_5</sub>, O<sub>73</sub>, O<sub>18</sub>, O<sub>11\_5</sub>, O<sub>83</sub>, O<sub>12\_3</sub>, O<sub>74</sub>, O<sub>23\_3</sub>, O<sub>17\_3</sub>, O<sub>95</sub>, O<sub>20\_10</sub>, O<sub>19</sub>, O<sub>17\_4</sub>, O<sub>26</sub>, O<sub>22\_4</sub>).

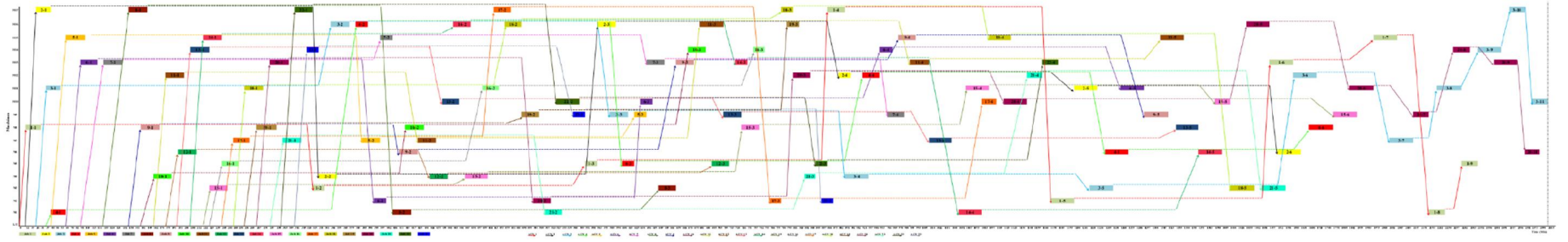


Figure 4.34: Operations' sequence before optimization – testbed 4

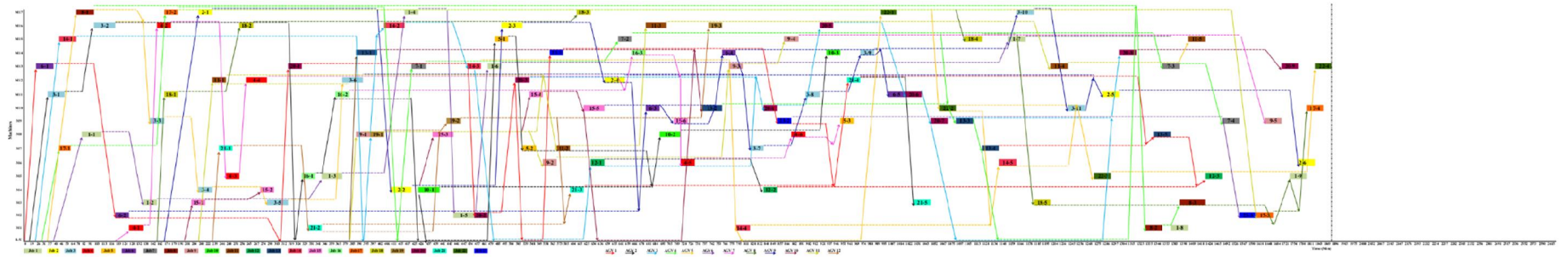


Figure 4.35: Operations' sequence after optimization by HGP2 – testbed 4

#### 4.4.5.2 AGVs' Battery Charge

To elaborate the effect of optimization on the AGVs' battery charge, Figure 4.36 and 4.37 are drawn to present the before and after optimization status. The level of consumed battery charge using each AGV both before and after the optimization and using all the EAs is shown in Figure 4.36, and similarly Figure 4.37 demonstrates the utilization of AGVs' battery charge. Similar to the other three testbeds, after the optimization using all the EAs, the number of AGVs was decreased and the batteries of omitted AGVs were saved, so the overall AGVs' battery charge utility were optimized to perform more jobs with one AGV. Therefore, the scheduling model and the EAs were also successful in this large-sized testbed to create a higher battery charge utility level compared with prior-optimization status (Figure 4.37).

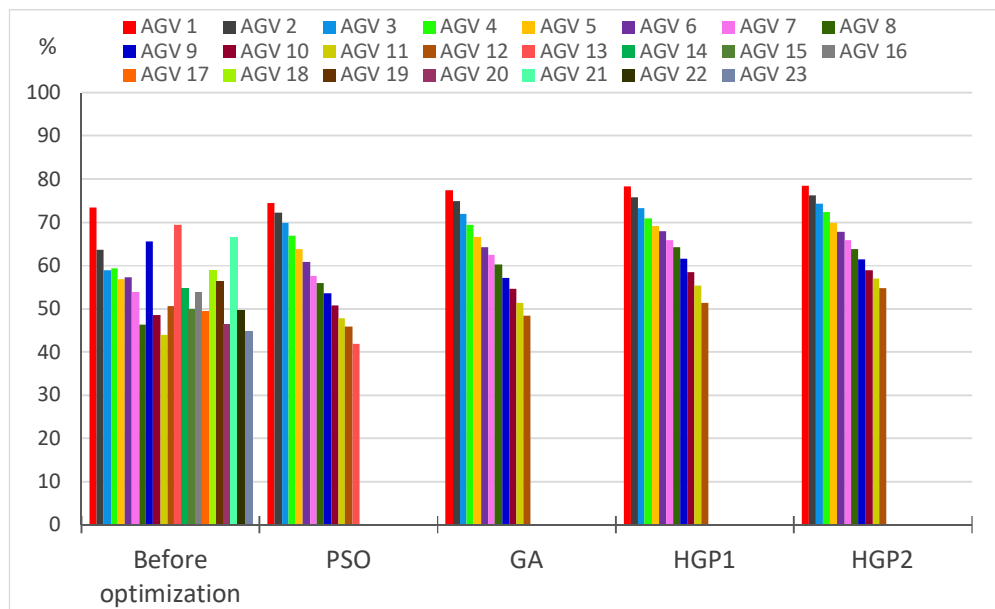


Figure 4.36: AGVs' battery charge consumption before and after optimization



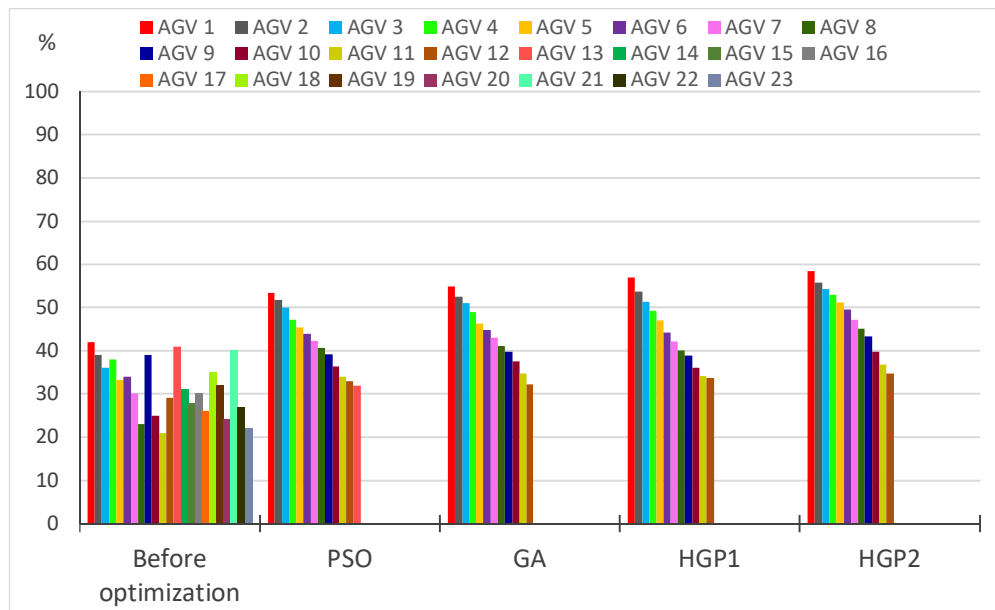


Figure 4.37: Battery charge utilization, before and after optimization – testbed 4

#### 4.4.5.3 AGVs' Specifications/Behavior

In Figure 4.38, the AGVs total running time prior to the optimization is lower than that of after optimization shown in Figure 4.39. It was discussed in previous testbeds that after the optimization the number of AGVs have been decreased and fewer AGVs would undertake the same workload, therefore the total running time of AGVs would increase after the optimization. Having fewer AGVs to do the same workload would definitely lessen the idle time of AGVs after the optimization. With such changes in the scheduling of AGVs, makespan would be shorter than that of before optimization. The above statements are realized in Figure 4.38 and 4.39 confirming the applicability and effectiveness of the optimization model and EAs in large size testbeds.

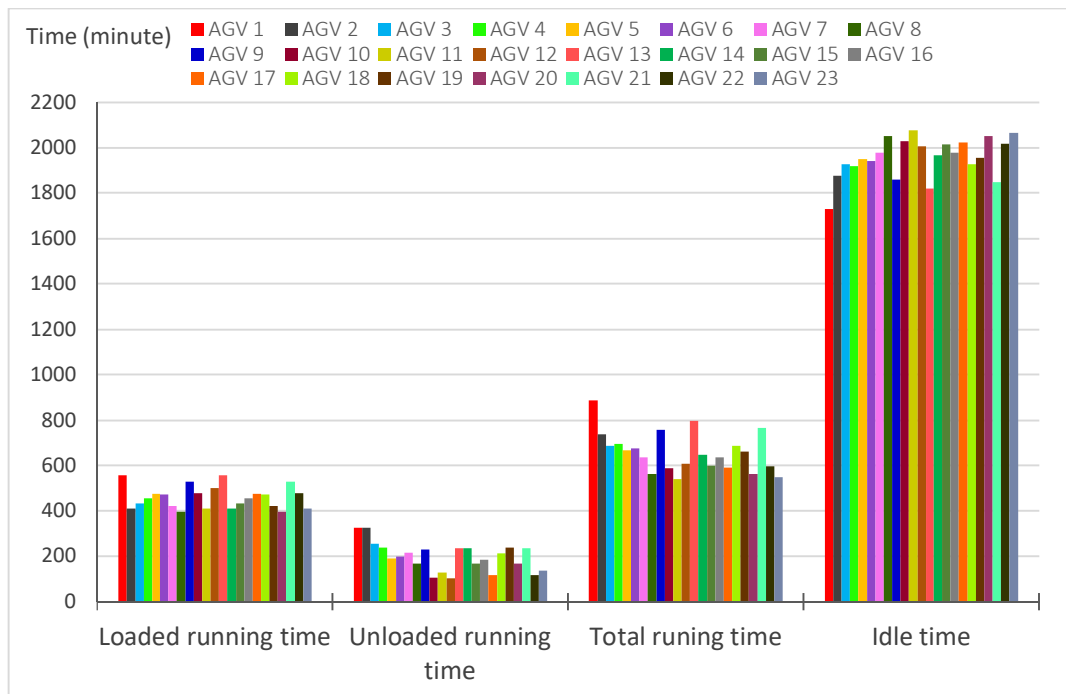


Figure 4.38: AGVs specification before optimization

After optimization, the makespan, number of AGVs and their idle time have been reduced as it is shown in Figure 4.39.

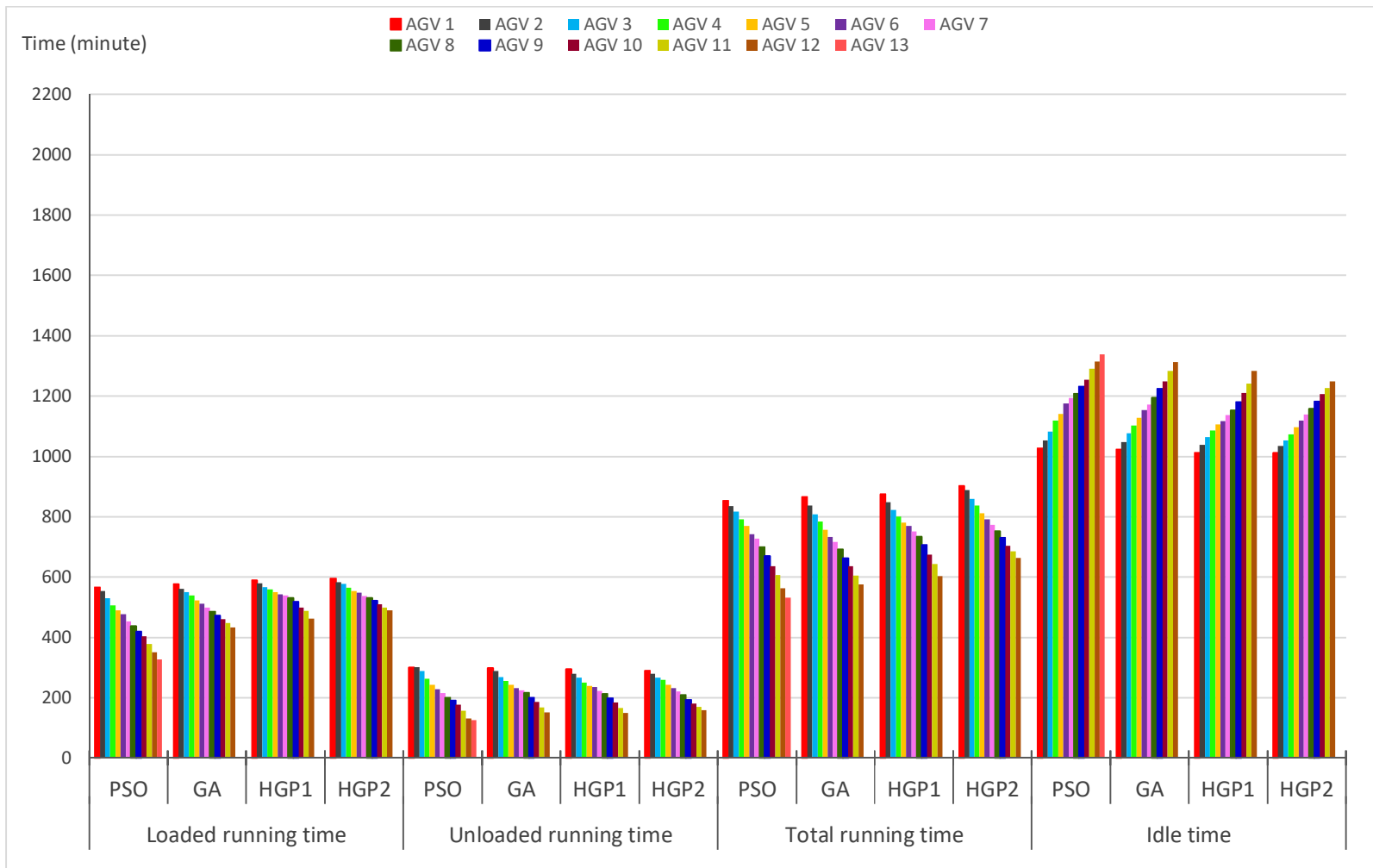


Figure 4.39: AGVs' specification after optimization

As it was shown earlier in Figure 4.36, the level of consumed battery charge decreased after the optimization, so that the AGVs' operation efficiency was consequently improved. Figure 4.40 demonstrates the enhanced efficiency of AGVs' operation after the optimization compared with prior optimization status. AGVs were deployed with no intention to use their highest potential and the AGV number one, nine, thirteen, and number twenty-one showed the highest operation efficiency before the optimization. After optimization, based on the scheduling model designed, tasks are sequentially appointed to AGVs based on their numbers order, so that AGV number one would have the highest operation efficiency level. This rule is to use the best potential of the first AGVs to avoid addition of extra AGVs and expenses involved. Following this strategy, Figures 4.40 illustrates the highest efficiency obtained by AGV1 and the lowest level by AGV12.

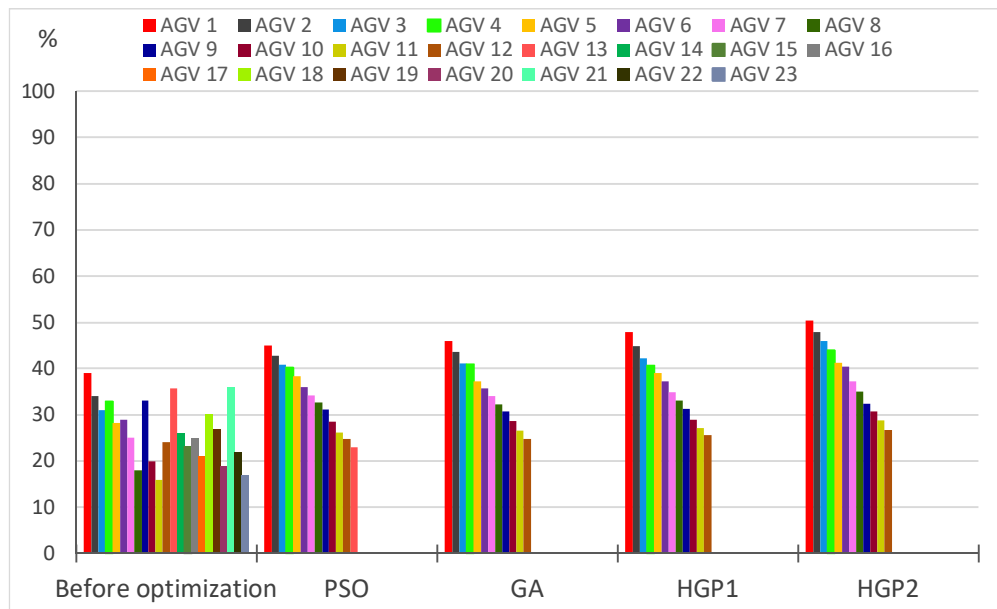


Figure 4.40: AGVs' operation efficiency before and after optimization

#### 4.4.6 Testbed-size Effect on Model and EAs

To provide a comparison basis among testbeds for the developed EAs, their best performances at all the testbeds are shown in Figure 4.41.

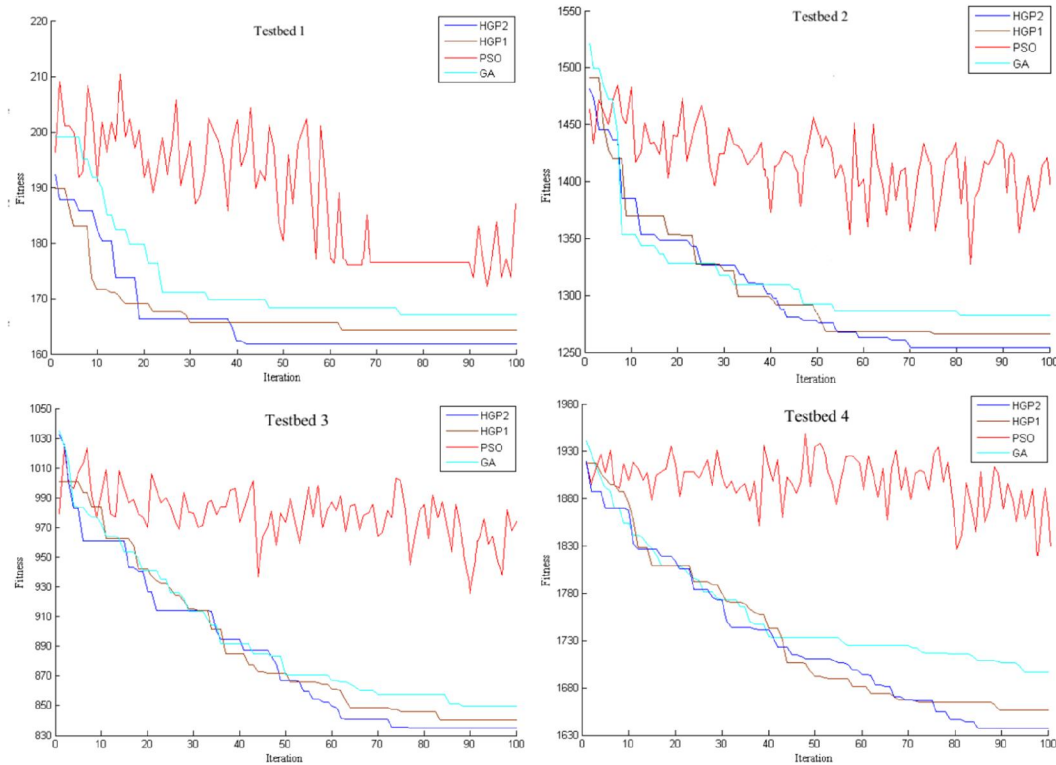


Figure 4.41: Best performance of the four algorithms at four testbeds

Figure 4.41 demonstrates the performance of all the algorithms in all the testbeds, in which a similar trend in the behaviour of the four EAs is evident. Increasing the problem size did not influence the response pattern of the studied EAs, and it only made the algorithms to converge at a higher iteration number. Enlargement of the testbed size imposed higher degree of complexity and difficulty to the algorithms for finding the optimum result. By increasing the problem size, the possible results space was widened which required more iteration till finding the optimum result.

As it is shown in Figure 4.41, in testbed 1, in the first 20 iterations HGP1 had the better result, but after 20<sup>th</sup> iteration HGP2 outperformed the other EAs. In testbed 2, in the

beginning, GA showed a fast convergence capability better than the other EAs, but HGP2 outpaced all EAs eventually. In testbed 3 and 4, because of the increment in problem size all the EAs converged at a higher iteration number compared with other testbeds, and HGP2 showed to be the best EA from the early iterations. It is noteworthy that, testbed 3 was inclusive of more machines, jobs, and operations than testbed 2, but its fitness function showed a lower makespan than testbed 2, which it is attributed to the shorter two-way distances and the shorter processing time of operations in Testbed 3.

In addition, in testbed 4 which is a large-sized testbed, the difference between HGP2's best result and the other EAs was heightened compared with that in other testbeds. It shows that in the bigger and more complicated problems HGP2 performs even better compared with smaller testbeds.

As it is shown in Table 4.14, the standard deviation of the best results of each EA followed the same pattern in all the testbeds, in which HGP2 had the smallest standard deviation and PSO had the largest one.

Table 4.14: Test results of optimization algorithms at four testbeds for hundred runs

	Testbed	PSO	GA	HGP1	HGP2
Best (min) fitness value	1	172.56	167.08	164.94	161.75
	2	1327.29	1282.63	1272.22	1253.96
	3	936.82	849.48	840.15	834.82
	4	1635.33	1573.5	1555.5	1542.83
Standard deviation of best (min) results	1	3.41	2.38	2.24	1.89
	2	18.87	18.2	17.06	15.48
	3	14.5	12.07	10.63	7.38
	4	29.52	26.62	17.31	14.53
Computational time (second)	1	51.11	49.61	54.38	48.5
	2	151.29	150.12	159.63	149.74
	3	170.29	189.74	176.63	187.12
	4	391.29	389.32	401.42	387.65

Algorithms response pattern to the testbed size in terms of the computational time followed the same pattern in all the testbeds (as shown in Table 4.14). Increasing the testbed size increased the computational time of all algorithms. It is clear that the bigger

problems require more time to process and find the optimum solution. However, in all the testbeds, HGP2 was the fastest computing algorithm followed by GA, PSO, and HGP1 respectively.

It was reported in the previous sections that in the bigger testbeds, which the role of AGVs battery consideration criterion is emboldened, as satisfactory results as in small-sized testbeds were obtained confirming the model performance independency of problem size. Similar patterns of battery charge consumption and battery charge utilization were observed at any of the four testbeds. AGVs specifications—running time (loading + unloading), and idle time—also showed the same response pattern when applying the model to the four different testbeds. In other words, testbed size did not show any significant impact on the model performance.

#### **4.4.7 EAs inter-comparison**

Overall, the four algorithms were proved successful in decreasing the makespan and required number of AGVs, and consequently improving the AGVs' battery charge utilization and AGVs' operation efficiency in all the testbeds. However, both the developed hybrids of HGP1 and HGP2 outperformed the GA and PSO. Literature had also highlighted the effectiveness of hybrid GA-PSO in solving scheduling problems and its superiority against the constituting algorithms (Jamrus et al., 2013; Kaveh & Malakouti Rad, 2010; Samuel & Rajan, 2015; Tang et al., 2010). The above conclusion, with regard to the present study, can be explained by the nature of the operators employed in the algorithms. In HGP1, benefiting from selection, crossover, and mutation operators, the population diversity would increase and facilitate finding new solution spaces and escaping the possible local optima (Dong et al., 2012; Settles & Soule, 2005). Therefore, such qualities empower HGP1 to outperform GA and PSO algorithms.

Comparing the two developed hybrids, it was seen that HGP2 surpassed HGP1 and converged at a faster rate to a lower value—in all the testbeds. The potential to improve upon HGP1 general hybridization style was explored by applying different elitism and selection approaches, and a different way of generating new population to the integration style. Rudolph (1999), Zitzler et al. (2000), Kumar et al. (2011), and Cao et al. (2016) had also utilized the elitism and proved its capabilities in fast convergence and computational time in multi-objective evolutionary algorithms. Another highlighting point in HGP2 structure was related to its population selection criteria that by comparing the generated population of two subsequent iterations, the best population was extracted. Choosing the best population at every iteration increased the chance of finding a better solution in less time (Jiang et al., 2017; Yu et al., 2015). The third distinguished characteristic in HGP2 was its higher exploitation and exploration capabilities. In HGP2, one half of the new population was made only by PSO operators and another half was made by applying both the PSO and GA operators. Therefore, in comparison with HGP1, its exploration and exploitation capabilities were improved (Cao et al., 2016; Soleimani & Kannan, 2015; J. Wang et al., 2016).

In hybrids, sometimes, adding and mixing options to create variety in the solutions in hope of better results, may cause complexity and increase the computational time like the HGP1 in this study. However, sometimes in hybrid cases such as HGP2 that characters like elitism are applied, as the worst solutions would diminish and less time would be spent on unsuitable solutions, the computational time may decrease (Yu & Gen, 2010a).

Standard deviation is another factor for EAs' performance comparison that provides important information on the existence of genetic redundancy in the population. Low standard deviation explains that the individuals have similar performances and probably this is due to the uniformity in chromosomes and particles (Mezura-Montes & Coello



Coello, 2004). As Kruse et al. (2013) mentioned, high values of standard deviation makes the algorithm to focus on exploration and low values of standard deviation makes it to focus on exploitation of search space (Črepinšek et al., 2013). HGP2 had the smaller standard deviation of the best results which indicates the close spread of the HGP2's results nearby/around the best results' mean value (Veček et al., 2014).

The model optimized using HGP2 showed to have higher AGV battery charge utilization compared with other EAs. It can be explained by the lower makespan obtained using HGP2, in which in such condition AGVs would perform their tasks in a shorter time. Thus, when AGVs battery charge utilization is increased, consequently their operation efficiency would be increased as it was seen in the four testbeds. Overall, from the findings of this research, apart from introducing the HGP2 configuration as a proper hybrid algorithm for similar problems, application of the hybrid GA-PSO in scheduling studies is affirmed to be more effective than its constituting EAs.

#### 4.5 Validation of the Optimization Model

To validate the proposed model, two layouts from Bilge and Ulusoy's study (Bilge & Ulusoy, 1995) were employed here and are shown in Figures 4.42 and 4.43.

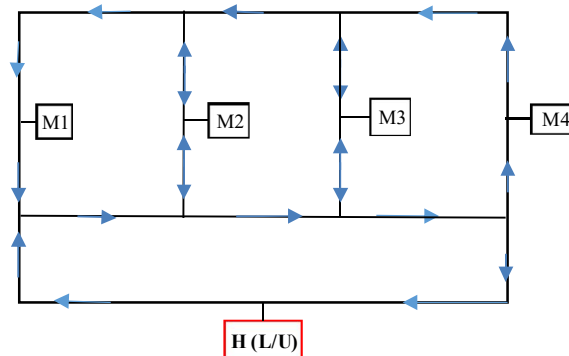


Figure 4.42: Layout 1 (Bilge & Ulusoy, 1995)

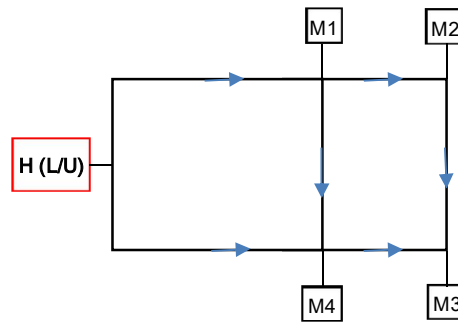


Figure 4.43: layout 2 (Bilge & Ulusoy, 1995)

Layouts' travel times are shown in Tables 4.15, 4.16. Data of ten job sets have been used which their details are represented in Table 4.17.

Table 4.15: Travel time (minutes) among L/U and machines – layout 1 (Bilge & Ulusoy, 1995)

	L/U	M <sub>1</sub>	M <sub>2</sub>	M <sub>3</sub>	M <sub>4</sub>
L/U	0	6	8	10	12
M <sub>1</sub>	15	0	2	5	7
M <sub>2</sub>	13	18	0	3	5
M <sub>3</sub>	10	15	17	0	2
M <sub>4</sub>	8	13	15	18	0

Table 4.16: Travel time (minutes) among L/U and machines – layout 2 (Bilge & Ulusoy, 1995)

	L/U	M <sub>1</sub>	M <sub>2</sub>	M <sub>3</sub>	M <sub>4</sub>
L/U	0	4	6	8	6
M <sub>1</sub>	6	0	2	4	2
M <sub>2</sub>	8	12	0	2	4
M <sub>3</sub>	6	10	12	0	2
M <sub>4</sub>	4	8	10	12	0

Table 4.17: Processing time (minutes) of operations on the machines (Bilge & Ulusoy, 1995)

Job set 1	Job number	1 1 1	2 2 2	3 3 3	4 4	5 5			
	Operation	1 2 3	1 2 3	1 2 3	1 2	1 2			
	Machine	M <sub>1</sub> M <sub>2</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>3</sub> M <sub>2</sub>	M <sub>3</sub> M <sub>4</sub> M <sub>1</sub>	M <sub>4</sub> M <sub>2</sub>	M <sub>3</sub> M <sub>1</sub>			
	Operation time	8 16 12	20 10 18	12 8 15	14 18	10 15			
Job set 2	Job number	1 1	2 2	3 3	4 4 4	5 5 5	6 6 6		
	Operation	1 2	1 2	1 2	1 2 2	1 2 3	1 2 3		
	Machine	M <sub>1</sub> M <sub>4</sub>	M <sub>2</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>3</sub>	M <sub>2</sub> M <sub>3</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>2</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>2</sub> M <sub>3</sub>		
	Operation time	10 18	10 18	10 20	10 15 12	10 15 12	10 15 12		
Job set 3	Job number	1 1	2 2	3 3	4 4	5 5 5 5	6 6 6 6		
	Operation	1 2	1 2	1 2	1 2	1 2 3 4	1 2 3 4		
	Machine	M <sub>1</sub> M <sub>3</sub>	M <sub>2</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>2</sub>	M <sub>3</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>2</sub> M <sub>3</sub> M <sub>4</sub>	M <sub>2</sub> M <sub>3</sub> M <sub>4</sub> M <sub>1</sub>		
	Operation time	16 15	18 15	20 10	15 10	8 10 15 17	10 15 8 15		
Job set 4	Job number	1 1 1	2 2 2	3 3 3 3	4 4 4 4	5 5 5 5 5			
	Operation	1 2 3	1 2 3	1 2 3 4	1 2 3 4	1 2 3 4 5			
	Machine	M <sub>4</sub> M <sub>1</sub> M <sub>2</sub>	M <sub>3</sub> M <sub>2</sub> M <sub>4</sub>	M <sub>2</sub> M <sub>3</sub> M <sub>1</sub> M <sub>3</sub>	M <sub>2</sub> M <sub>4</sub> M <sub>1</sub> M <sub>2</sub>	M <sub>1</sub> M <sub>2</sub> M <sub>4</sub> M <sub>2</sub> M <sub>3</sub>			
	Operation time	11 10 7	12 10 8	7 10 9 8	7 8 12 6	9 7 8 10 8			
Job set 5	Job number	1 1 1	2 2 2	3 3 3	4 4	5 5			
	Operation	1 2 3	1 2 3	1 2 3	1 2	1 2			
	Machine	M <sub>1</sub> M <sub>2</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>3</sub> M <sub>2</sub>	M <sub>3</sub> M <sub>4</sub> M <sub>1</sub>	M <sub>4</sub> M <sub>2</sub>	M <sub>3</sub> M <sub>1</sub>			
	Operation time	6 12 9	18 6 15	9 3 12	6 15	3 9			
Job set 6	Job number	1 1 1	2 2 2	3 3 3	4 4 4	5 5 5	6 6 6		
	Operation	1 2 3	1 2 3	1 2 3	1 2 3	1 2 3	1 2 3		
	Machine	M <sub>1</sub> M <sub>2</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>2</sub> M <sub>4</sub>	M <sub>2</sub> M <sub>3</sub> M <sub>4</sub>	M <sub>2</sub> M <sub>3</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>3</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>3</sub> M <sub>4</sub>		
	Operation time	9 11 7	19 20 13	14 20 9	14 20 9	11 16 8	10 12 10		
Job set 7	Job number	1 1	2 2	3 3	4 4	5 5	6 6 6	7 7 7	8 8 8
	Operation	1 2	1 2	1 2	1 2	1 2	1 2 3	1 2 3	1 2 3
	Machine	M <sub>1</sub> M <sub>4</sub>	M <sub>2</sub> M <sub>4</sub>	M <sub>2</sub> M <sub>4</sub>	M <sub>3</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>3</sub>	M <sub>2</sub> M <sub>1</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>2</sub> M <sub>3</sub>	M <sub>1</sub> M <sub>2</sub> M <sub>4</sub>
	Operation time	6 6	11 9	9 7	16 7	9 18	19 21 6	10 9 13	11 9 8
Job set 8	Job number	1 1 1	2 2 2	3 3 3	4 4 4	5 5 5 5	6 6 6 6		
	Operation	1 2 3	1 2 3	1 2 3	1 2 3	1 2 3 4	1 2 3 4		
	Machine	M <sub>2</sub> M <sub>3</sub> M <sub>4</sub>	M <sub>2</sub> M <sub>3</sub> M <sub>4</sub>	M <sub>2</sub> M <sub>3</sub> M <sub>4</sub>	M <sub>2</sub> M <sub>3</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>2</sub> M <sub>3</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>2</sub> M <sub>3</sub> M <sub>4</sub>		
	Operation time	12 21 11	12 21 11	12 21 11	12 21 11	10 14 18 9	10 14 18 9		
Job set 9	Job number	1 1 1 1	2 2 2	3 3 3	4 4 4	5 5 5 5			
	Operation	1 2 3 4	1 2 3	1 2 3	1 2 3	1 2 3 4			
	Machine	M <sub>3</sub> M <sub>1</sub> M <sub>2</sub> M <sub>4</sub>	M <sub>3</sub> M <sub>2</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>2</sub> M <sub>4</sub>	M <sub>2</sub> M <sub>3</sub> M <sub>4</sub>	M <sub>3</sub> M <sub>1</sub> M <sub>2</sub> M <sub>4</sub>			
	Operation time	9 12 9 6	16 11 9	21 18 7	20 22 11	14 16 13 9			
Job set 10	Job number	1 1 1 1	2 2 2	3 3 3 3	4 4 4	5 5 5	6 6 6 6		
	Operation	1 2 3 4	1 2 3	1 2 3 4	1 2 3	1 2 3	1 2 3 4		
	Machine	M <sub>1</sub> M <sub>3</sub> M <sub>2</sub> M <sub>4</sub>	M <sub>2</sub> M <sub>3</sub> M <sub>4</sub>	M <sub>3</sub> M <sub>2</sub> M <sub>1</sub> M <sub>4</sub>	M <sub>2</sub> M <sub>3</sub> M <sub>4</sub>	M <sub>1</sub> M <sub>3</sub> M <sub>4</sub>	M <sub>2</sub> M <sub>1</sub> M <sub>3</sub> M <sub>4</sub>		
	Operation time	11 19 16 13	21 16 14	8 10 14 9	13 20 10	9 16 18	19 21 11 15		

Table 4.18 shows the comparison of the makespan of the result of twenty test problems using a combination of two different facility layout from HGP2 along with the benchmark

results of the algorithm in the literature that used the same data. The problems were solved by Bilge and Ulusoy (1995), Ulusoy et al. (1997), Abdelmaguid et al. (2004), Reddy and Rao (2006), Deroussi et al. (2008), Gnanavel Babu et al. (2010), and Zheng et al. (2014) and the test results are shown in Table 4.18. “L” is abbreviation for layout, first number shows the layout number and second number shows the job set.

Table 4.18: Comparison of makespan results

Test No.	B95	U97	A04	R06	D08	B10	Z14	U15	HGP2
L1-1	96	96	96	96	96	94	96	80	96
L1-2	105	104	102	100	102	108	100	88	101
L1-3	105	105	99	99	99	87	99	119	98
L1-4	118	116	112	112	112	85	112	94	118
L1-5	89	87	87	87	87	80	87	66	78
L1-6	120	121	118	118	118	114	118	130	113
L1-7	119	118	115	111	111	90	111	86	108
L1-8	161	152	161	161	161	145	161	146	145
L1-9	120	117	118	116	116	115	116	121	116
L1-10	153	150	147	147	147	121	146	152	140
L2-1	82	82	82	82	82	88	82	79	82
L2-2	80	76	76	76	76	86	76	84	75
L2-3	88	85	85	85	85	74	85	116	85
L2-4	93	88	88	87	87	74	87	92	74
L2-5	69	69	69	69	69	76	69	65	69
L2-6	100	98	98	98	98	92	98	128	95
L2-7	90	85	79	79	79	70	79	83	75
L2-8	151	142	151	151	151	123	151	145	140
L2-9	104	102	104	102	102	95	102	127	102
L2-10	139	137	136	135	135	113	135	149	150

B95-(Bilge & Ulusoy, 1995), U97-(Ulusoy et al., 1997), A04-(Abdelmaguid et al., 2004), R06-(Reddy & Rao, 2006), D08-(Deroussi et al., 2008), B10-(Gnanavel Babu et al., 2010), Z14-(Zheng et al., 2014), (Umar et al., 2015)

It is observed from the above results that HGP2 performed better than all the other previous algorithms in the four test problems (L1-5, L1-6, L2-2, L2-7), it was worse in two test problems (L1-4, L2-10) and its results were in the range of others' result for the remaining test problems.

## 4.6 Validation of Optimization Result

The result of optimization was validated through simulation of one of the medium-sized testbeds (testbed 2 as a representative for all the testbed sizes and properties). FlexSim software version 2016, update 2 (16.2.2) used as the simulation software. The experiments were run on a desktop computer with a 2.80 GHz processor and 4 GB RAM.

### 4.6.1 Layout Set up

The first step in simulation by FlexSim is to set up the layout. The layout was built based on the defined distances among machines and home and Figure 4.44 displays a part of the testbed layout. In layout design, after defining the machine places and distances, there should be a place for parts to enter the system, and also a place to distribute the parts. In the simulated model, each part enters the system through ‘Source’ and the sequence defined in the source represents the job sequence. ‘Sink’ acts as home ( $H$ ), the place that all the parts are distributed from. “Sink” and “Source” are shown by a red rectangular in Figure 4.44. Processors are used as machines (the green rectangular) and a ‘Queue’ object which is shown by the blue rectangular (named by Q1, Q2, ...) placed after each machine acts as a space for the processed parts waiting for the AGV pick-up. The yellow vehicles (shown by yellow rectangular in Figure 4.44) are the AGVs that collect the parts/products either from the sink resource or machines and deliver as scheduled. The queue and machine share the same control point to ensure the accuracy of the calculation. The starting point for all the AGVs is at the control point connected to  $H$ , and the travelling route is calculated by the software algorithm based on the layout data.

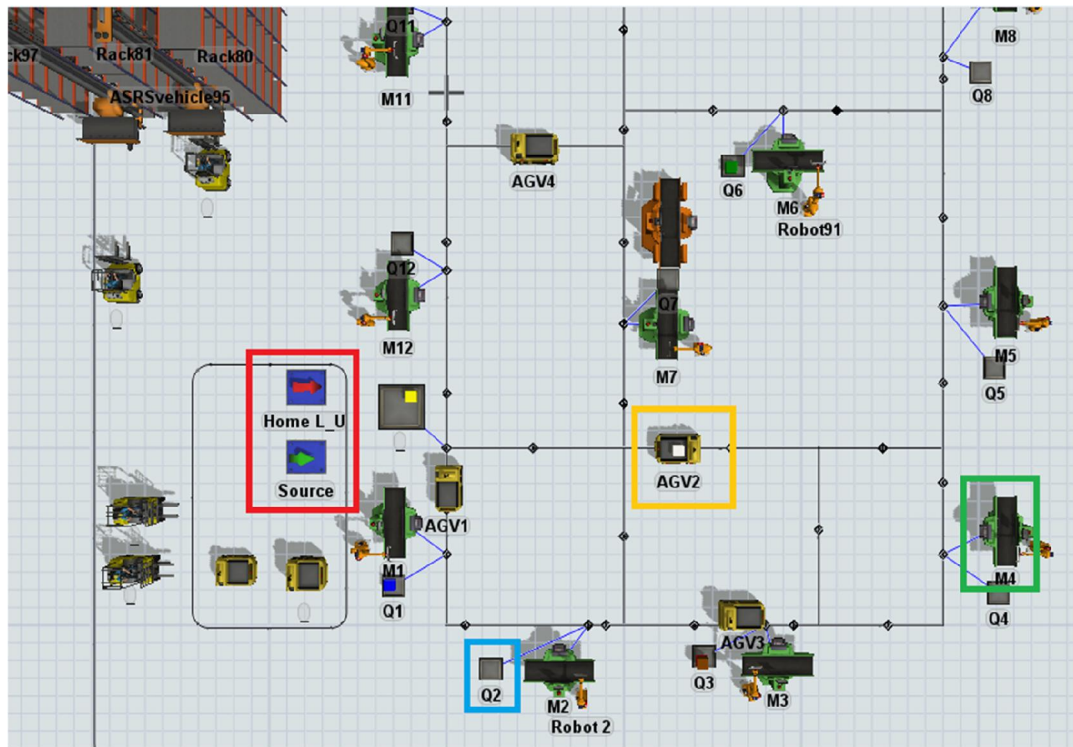


Figure 4.44: Sink, source, AGV, and machines in the simulated model environment

#### 4.6.2 Model's Rules and Information Entry to the FlexSim Database

After setting up the layout, the information and rules such as number of jobs, operations, processing times of each operation, operations sequence, and details of assigned AGVs to each operation should be entered to the simulation environment. In FlexSim, assigned AGVs to each part are controlled by a table named “job”. Figure 4.45 shows a part of the job table, which its sequence is based on Figure 4.11.

job				Rows
Job_type	Destination	AGV		36
Row 2	4.0	1.0	1.0	
Row 3	1.0	2.0	2.0	
Row 4	4.0	7.0	1.0	
Row 5	4.0	5.0	1.0	
Row 6	2.0	3.0	3.0	
Row 7	4.0	12.0	1.0	
Row 8	6.0	3.0	4.0	
Row 9	3.0	1.0	5.0	
Row 10	1.0	6.0	2.0	

Figure 4.45: Part of the “job table” in FlexSim

The first column in Table 4.45 is the job number and the repetition of that represents the operations, the second column shows the machine number. Third column shows the assigned AGV to an operation. For example, if the operation  $O_{12}$  on machin  $M_6$  is assigned to AGV2, then its presence in the Table 4.43 from the left would be “1” (second number 1 in the Job\_type’s column from the top), “6”, and “2”.

Each job will arrive at the system through “Source”. Figure 4.46 shows the properties of the "Source". The arrival sequence represents the sequence of jobs arrival. The number of arrival is 6 as there are 6 jobs.

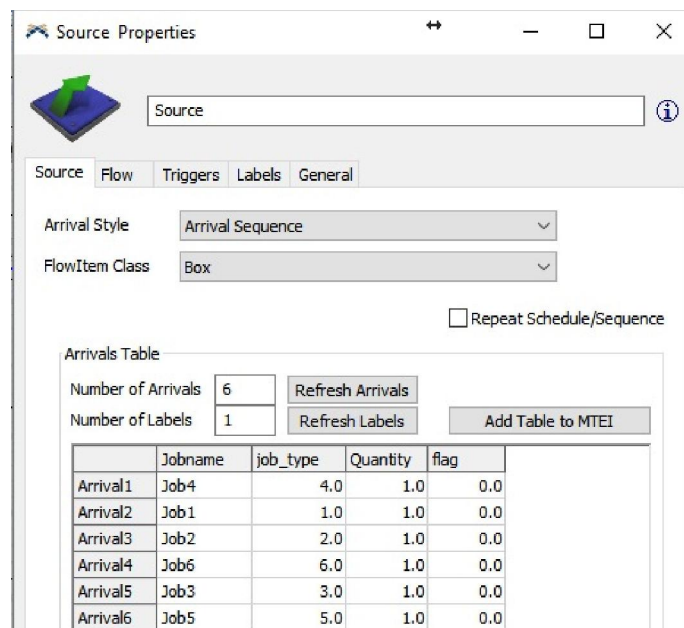


Figure 4.46: Source properties

The processing time (minute) for each job was based on Table 4.6. Information is presented in the table “Op\_Time” as in Figure 4.47 with rows and columns representing the jobs and the operations, respectively.

	Operation 1	Operation 2	Operation 3	Operation 4	Operation 5	Operation 6	Operation 7	Operation 8
Job 1	37.0	33.0	34.0	35.0	23.0	34.0	37.0	26.0
Job 2	23.0	26.0	27.0	25.0	34.0	23.0	0.0	0.0
Job 3	26.0	25.0	31.0	24.0	25.0	13.0	14.0	23.0
Job 4	16.0	11.0	23.0	34.0	25.0	13.0	0.0	0.0
Job 5	16.0	11.0	31.0	0.0	0.0	0.0	0.0	0.0
Job 6	26.0	31.0	23.0	24.0	35.0	0.0	0.0	0.0

Figure 4.47: Table of operation time of the example in FlexSim

Figure 4.48 shows the properties of a processor (machine). Setting of the operation times of machines is shown by the red arrow in Figure 4.48. In this step, by selecting the table “Op\_Time” from Figure 4.47 in the dropdown menu and choosing the correct job and operation number in the “Row” and “Column” sections, the correct processing time is located.

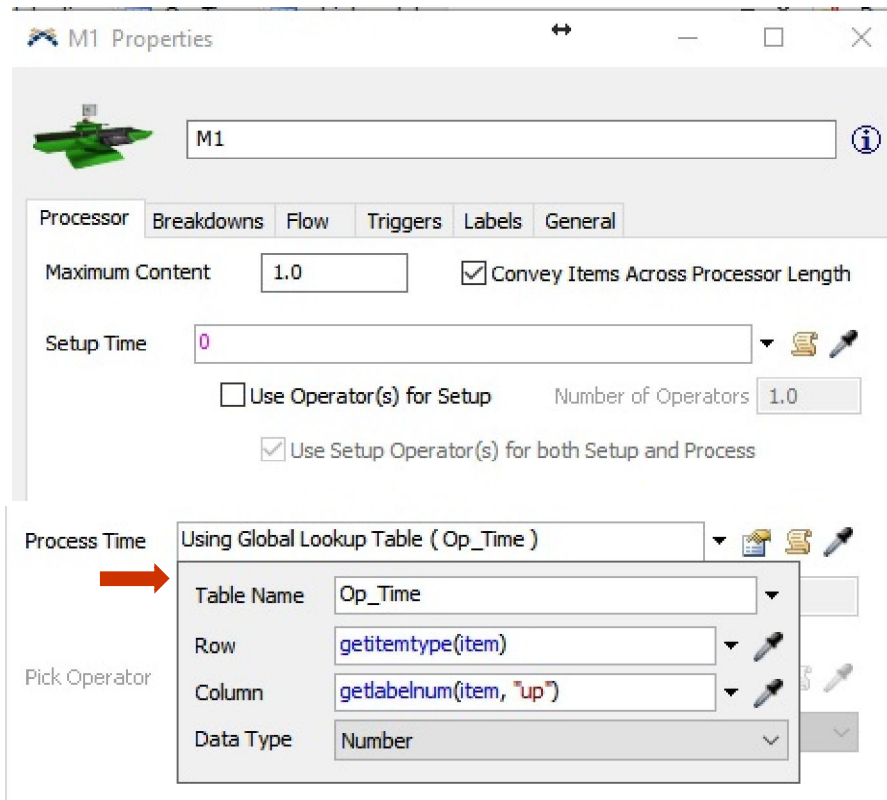
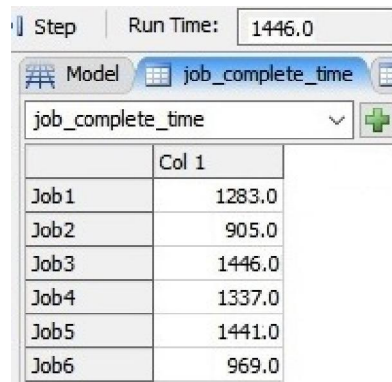


Figure 4.48: Machine properties



### 4.6.3 Simulation Result

After running the simulated model, all the jobs completion time werereported in “job\_completion\_time” table of the software shown in Figure 4.49. Since makespan is the completion time of all jobs, the biggest number in the table represents the makespan, which is related to job 3 being the last job to be completed (Figure 4.49).



Job	Col 1
Job1	1283.0
Job2	905.0
Job3	1446.0
Job4	1337.0
Job5	1441.0
Job6	969.0

Figure 4.49: Completion time for each job

There are other figures reporting everything about the simulated model, after finishing the jobs. Figure 4.50 shows the time that every part has spent for a particular work classifying as waiting time, processing time and travelling time of the parts in the model. Figure 4.51 is the figure that depicts the time for each buffering queue spent on collecting, releasing and being empty.

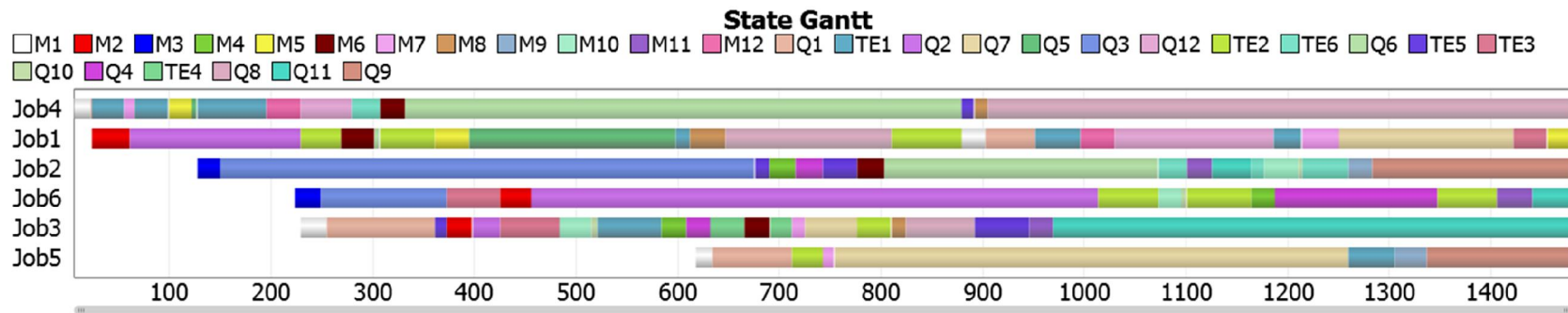


Figure 4.50: Waiting time, processing time and travelling time of the goods

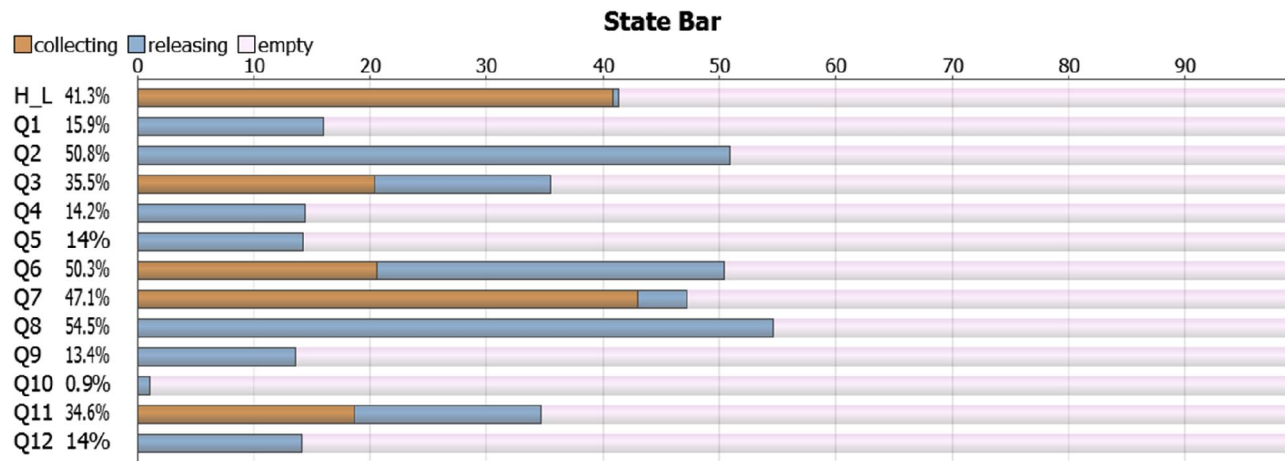


Figure 4.51: Buffering queues' time on collecting, releasing and being empty

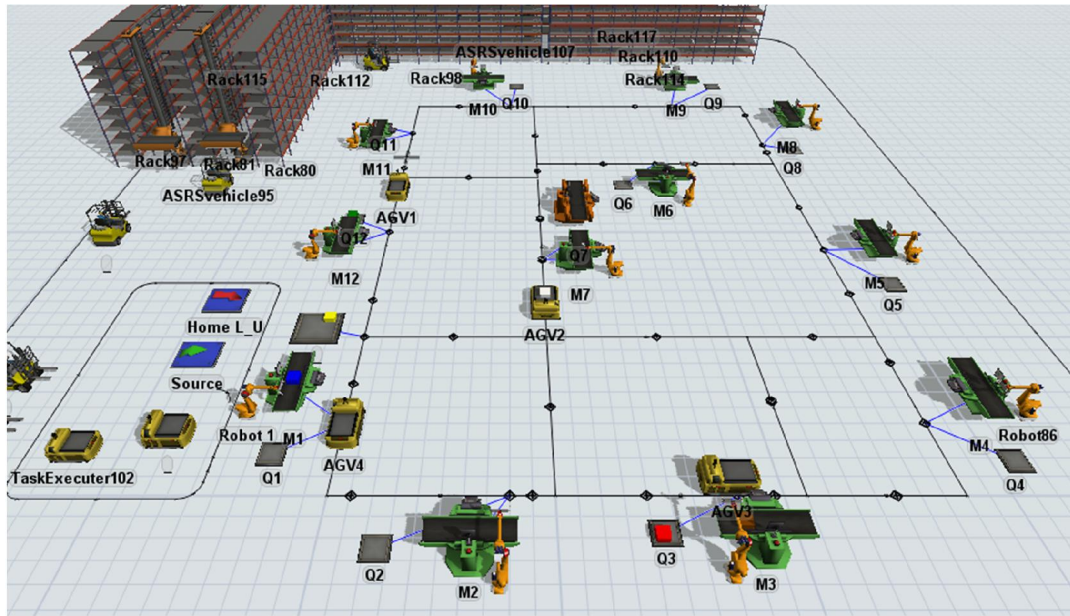


Figure 4.52: Simulation environment, when the model is running

Figure 4.52 shows a scene from the simulation environment, when the model was running and AGVs were functioning. Using testbed 2 configuration and applying its optimized sequence by HGP2 for the simulation in FlexSim software, it was found that the simulation results conform to optimization results and similar makespan magnitude and number of AGVs were obtained. Therefore, model simulation in FlexSim software proved the validity of the optimization result.

#### 4.7 Summary

After accomplishing the model and algorithms development step, the optimal parameters set of each EA was found using Brute-force method. Then, algorithms were applied to four testbeds with different sizes (1 small, 2 mediums, and 1 large) and their performances were studied. The algorithms developed were successful in decreasing the makespan and the required number of AGVs in all the testbeds. Through the superior scheduling of AGVs, after the optimization, AGVs' idle time was decreased and their battery utilization level was improved.

HGP2 outperformed the other three algorithms and obtained a better fitness value at a faster convergence rate at all the testbeds. Comparing the computational time of the four EAs, HGP1 had the longest computational time, while HGP2 had the smallest one and the other two algorithms varied closely. Concerning the best results obtained by each EA, HGP2 showed the smallest standard deviation followed by HGP1, GA, and PSO, respectively. This study, similar to the previous studies, confirmed the superiority of the hybrids of GA and PSO over each of them being individually applied. However, selection of the right operators and integration approach in hybridization of the EAs was proved to be highly influential on the performance of the constructed hybrid. Adding selection, crossover, and mutation operators into the hybrids increased the diversity of the population and facilitated finding new solution spaces and escaping the possible local optima. In addition, employing elitism, different kind of parents' selection, and having a new population selection strategy (half from GA and half from PSO) improved the HGP2's performance.

The testbed size effect on the EAs performance was explored next. However, it did not show any impact on the response pattern of the studied EAs and only their convergence was postponed to higher iterations at large-sized testbeds. Then, for model validation, benchmarking was used and the obtained results proved the feasibility and validity of the model. Therefore, this model can be used as a reference for similar studies in AGV scheduling context. Final, in order to validate the optimization result, a simulation practice based on the testbed 2 was performed using the FlexSim software.

## CHAPTER 5: CONCLUSIONS

### 5.1 Research Summary

The prognostics, concluding from the world trade growth, show the ever-increasing application of AGVs in the industry. The process of allocating AGVs to tasks, taking into account the costs and time of operations—so called scheduling—is the challenge to administer in this context. Efficient scheduling, which is the resultant of application of a model with high efficiency and fast throughput that shortens the production time and decreases the costs in an FMS, is pursued in this study and many others. Review of the literature on FMS, AGV, and scheduling discourses showed that every study analyzes the impacts of a different criteria set on FMS performance. The criteria and scheduling models were developed mainly based on a specific hypothesis drew by every researcher. To broaden the scheduling knowledge, this study developed a multi-objective scheduling optimization model with a new set of criteria of makespan minimization and minimization of AGVs number with their battery charge consideration involved.

After determining the criteria set of the multi-objective model, based on the reviewed literature, a mathematical model representing the scheduling model was developed and presented in section 3.3. With scheduling problems being characterized as NP-hard problems, evolutionary algorithms (EAs) were employed to find an optimized solution. Following the “no free lunch” theory, many studies had proposed the application of GA and PSO for scheduling problems. Therefore, GA and PSO algorithms were used and developed for the problem at hand. Literature had also highlighted that a hybrid of GA and PSO can be more effective than each of them being applied individually. Thus, taking advantage of GA and PSO unique capabilities, two hybrids (HGP1, HGP2) were developed. The difference between them relied on the approaches applied for the GA and PSO operators’ integration. Overall, four evolutionary algorithms (GA, PSO, HGP1, and HGP2) were developed for the problem and presented in section 3.4.

Following experts' suggestions, MATLAB software was used for the coding of the model and all the four algorithms. Performance of the model and algorithms were examined using numerical application of them at different testbeds. As there was a new criterion (AGVs' battery charge considerations) included in the model that distinguishes it from existing models, it was realized that the available testbeds were not suitable to the problem at hand both in terms of the size and the problem definition. Therefore, four new testbeds with different sizes (1 small, 2 mediums, and 1 large-sized) were defined in sections 4.4.2 to 4.4.5 to assess the model. Next, the model validity was tested through simulation in FlexSim software.

## **5.2 Conclusions**

Analysis of the four algorithms' result at different testbeds showed the optimization model functionality and the algorithms success in decreasing the makespan and the required number of AGVs. By employing the three defined criteria in the developed model, the number of utilized AGVs was decreased after the optimization. Therefore, the battery charge of omitted AGVs was preserved. In addition, the model was successful in optimizing the battery charge utilization of the remained/working AGVs in that the same number of AGVs were able to perform more jobs in less time using nearly the same or a bit higher battery charge. It indicates that the battery charge utilization has been enhanced using the model developed. Thus, the model appeared capable in reducing the FMS costs by decreasing the number of AGVs applied for the same volume of jobs.

The model was further scrutinized by studying the AGVs characteristics/behavior such as the total running time (loaded and unloaded), idle time, and AGVs' operation efficiency—before and after the optimization. It was found that after optimization, despite the small rise in AGVs' total running time (loaded and unloaded), the AGVs' idle time was reduced dramatically. With the reduction of idle time, the AGVs' operation

efficiency has been enhanced. This shows that the use of AGVs was more effective after the optimization, in which it is again a source of cost reduction in the FMS while the overall efficiency of the system is enhanced.

All the four algorithms found an optimized result, although HGP2 had a better performance compared with PSO, GA, and HGP1. The optimized model using HGP2 converged at a faster rate and to a lower value. Concerning the variation between worst and best results obtained by each EA, HGP2 was able to maintain a wide range of candidate solutions and provide more diversity compared with other EAs. Of the four EAs, HGP2 obtained the smallest standard deviation with respect to the best results generated, and the HGP2's capability in better convergence. HGP1, GA, and PSO were the next algorithm with respectively small standard deviations. Comparing algorithms in terms of computational time, it was found that HGP2 was the faster performing EA than GA, PSO and HGP1 respectively.

As discussed above, despite the superiority of HGP2 over HGP1, both the hybrids outperformed the GA, and PSO algorithms. In HGP1, selection, crossover, and mutation operators were integrated together and that increased the diversity of the population and facilitated discovering new solution spaces and escaping the possible local optima. In HGP2, incorporation of the elitism operator into the algorithm along with the innovative population selection process applied, and novel structure of integrating the operators of GA and PSO have escalated its exploration and exploitation capabilities and the convergence rate. Overall, apart from introducing the HGP2 as a suitable hybrid for similar problems; in line with the literature consensus on the excellence of hybrids over their constituents being individually applied, the present research also reaffirms this statement.

With regard to the testbed size impact on the model applicability and the algorithms performance, it was seen that the alteration of testbed size did not affect the algorithms performance pattern and their level of optimality when compared with one another. Increasing the testbed size postponed the convergence of all EAs to higher iterations, because enlargement of the testbed size imposed higher degree of complexity and difficulty to the algorithms for finding the optimum result. In the large-sized testbeds, the difference between HGP2 and HGP1 best results was increased compared to smaller sized testbeds. Following the same fashion, the difference between HGP2 and GA and PSO was also mounted in the large-sized testbed, which it indicates that HGP2 has been even more powerful in large-sized testbeds than in the small-sized problems.

To validate the model applicability, the proposed model has been validated through some well-known benchmarking problems with data of ten job sets on two layouts. Then, the results were compared with previous algorithms which used the same data for scheduling. For validating optimization result, the second testbed (medium-sized) was chosen to be simulated by FlexSim in section 4.5. The choice of second testbed for simulation was to avoid the unnecessary complexity driven by a big model application, and for a proper representation of the results, and to have a representative of all problem sizes. The obtained results were the same as that in the model runs in MATLAB. Thus, in line with the testbeds results, simulation using the FlexSim software has also proved the feasibility of the developed model. So that, the model developed in this study can be introduced as an efficient and competent model for similar scheduling tasks.

### **5.3 Future Research**

Based on the nature of the scheduling studies and similar to many researches in this area, some limitations as explained in chapter 3 were applied for the model development in the



present study. However, the following research potentials are recommended to stretch out the developed multi-objective model in this study:

- Considering AGVs and machines' breakdown

In case of an AGV/machine breakdown, the vehicle will be stranded in the path or the machine process will be stopped. It will block all the vehicles carrying out the P/D task. Considering such a criterion in the model to observe the breakdowns would improve the model in terms of the reality resemblance.

- Conflict-free routing

The developed model is assumed to be conflict-free, but in reality, conflicts may happen and it can be added to model to extend the study. Real time issues like traffic congestion and conflicts can also be considered.

- Developing different algorithms and hybrids for the model

Other algorithms and hybrids could be developed for the model and compared with the algorithms being used in this study.

## REFERENCES

- Abdelmaguid, T. F., et al. (2004). A hybrid GA/heuristic approach to the simultaneous scheduling of machines and automated guided vehicles. *International Journal of Production Research*, 42(2), 267-281.
- Agrawal, R., et al. (2012). Scheduling of a flexible job-shop using a multi-objective genetic algorithm. *Journal of Advances in Management Research*, 9(2), 178-188.
- AGV Kennis Instituut. (2015). General technology description of AGV-systems. Retrieved from [http://www.frog.nl/Oplossingen/AGV\\_Kennis\\_Instituut](http://www.frog.nl/Oplossingen/AGV_Kennis_Instituut).
- Aized, T. (2009). Modelling and performance maximization of an integrated automated guided vehicle system using coloured Petri net and response surface methods. *Computers & industrial engineering*, 57(3), 822-831.
- Akturk, M., & Yilmaz, H. (1996). Scheduling of automated guided vehicles in a decision making hierarchy. *International Journal of Production Research*, 34(2), 577-591.
- Al Theeb, N. A., & Alhwiti, T. (2014). *Solving single-machine weighted tardiness and total setup costs scheduling problem with sequence dependant setup times and sequence dependant setup cost using discrete particle swarm*. Paper presented at the Industrial and Systems Engineering Research Conference, Montreal, Canada.
- Albert, P., & Castagna, P. (1996). *Piloting of AGV systems with conflict management method*. Paper presented at the CESA'96 IMACS Multiconference on computational engineering in systems applications, Lille, France.
- Almada-Lobo, F. (2016). The Industry 4.0 revolution and the future of manufacturing execution systems (MES). *Journal of Innovation Management*, 3(4), 16-21.
- Andersson, M., et al. (2016). *Tuning of multiple parameter sets in evolutionary algorithms*. Paper presented at the Proceedings of the 2016 on Genetic and Evolutionary Computation Conference.
- Anwar, M. F., & Nagi, R. (1998). Integrated scheduling of material handling and manufacturing activities for just-in-time production of complex assemblies. *International Journal of Production Research*, 36(3), 653-681.
- Ariffin, M., et al. (2011). Automated guided vehicles scheduling optimization by fuzzy genetic algorithm.
- Aydemir, E., & Koruca, H. (2015). A new production scheduling module using priority-rule based genetic algorithm. *International Journal of Simulation Modelling (IJSIMM)*, 14(3).
- Aytug, H., et al. (2003). Use of genetic algorithms to solve production and operations management problems: a review. *International Journal of Production Research*, 41(17), 3955-4009.

- Azimi, P. (2011). Alleviating the collision states and fleet optimization by introducing a new generation of automated guided vehicle systems. *Modelling and Simulation in Engineering, 2011*, 2.
- Aziz, N. A. A., et al. (2011). *Particle swarm optimization for constrained and multiobjective problems: A brief review*. Paper presented at the International Proceedings of Economics Development & Research, Bali, Indonesia.
- Ba, L., et al. (2016). Modelling and simulation of a multi-resource flexible job-shop scheduling. *International Journal of Simulation Modelling (IJSIMM)*, 15(1).
- Badakhshian, M., et al. (2012). Performance optimization of simultaneous machine and automated guided vehicle scheduling using fuzzy logic controller based genetic algorithm. *International Journal of Physical Sciences*, 7(9), 1461-1471.
- Baker, K. R. (1995). *Elements of sequencing and scheduling*. Amos Tuck School of Business Administration, Dartmouth College, Hanover, United States.
- Beheshti, Z., & Shamsuddin, S. M. H. (2013). A review of population-based meta-heuristic algorithms. *Int. J. Adv. Soft Comput. Appl*, 5(1), 1-35.
- Berman, S., & Edan, Y. (2002). Decentralized autonomous AGV system for material handling. *International Journal of Production Research*, 40(15), 3995-4006.
- Biegel, J. E., & Davern, J. J. (1990). Genetic algorithms and job shop scheduling. *Computers & industrial engineering*, 19(1), 81-91.
- Bilge, Ü., & Ulusoy, G. (1995). A time window approach to simultaneous scheduling of machines and material handling system in an FMS. *Operations Research*, 43(6), 1058-1070.
- Bioportfolio. (2017). Automated guided vehicle market by type (unit load carrier, tow vehicle, pallet truck, assembly line vehicle), industry vertical (automotive, & others), application (transportation, distribution, & others), & geography - global forecast to 2020. Retrieved from <https://www.bioportfolio.co.uk/product/41497>.
- Blazewicz, J., et al. (1991). Scheduling tasks and vehicles in a flexible manufacturing system. *International Journal of Flexible Manufacturing Systems*, 4(1), 5-16.
- Cai, Q., et al. (2014). Multi-AGV scheduling optimization based on neuro-endocrine coordination mechanism. *International Journal on Smart Sensing and Intelligent Systems*, 7(4), 1613-1630.
- Cao, L., et al. (2016). A guiding evolutionary algorithm with greedy strategy for global optimization problems. *Computational intelligence and neuroscience, 2016*.
- Chandrasekaran, S., et al. (2007). *Multi-objective particle swarm optimization algorithm for scheduling in flowshops to minimize makespan, total flowtime and completion time variance*. Paper presented at the IEEE Congress on Evolutionary Computation. CEC 2007.

- Chelouah, R., & Siarry, P. (2003). Genetic and Nelder–Mead algorithms hybridized for a more accurate global optimization of continuous multimimima functions. *European Journal of Operational Research*, 148(2), 335-348.
- Chen, C. L., et al. (1995). An application of genetic algorithms for flow shop problems. *European Journal of Operational Research*, 80(2), 389-396.
- Cheng, R., et al. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation. *Computers & industrial engineering*, 30(4), 983-997.
- Chudasama, C., et al. (2011). *Comparison of parents selection methods of genetic algorithm for TSP*. Paper presented at the International Conference on Computer Communication and Networks CSI-COMNET-2011, Proceedings.
- Črepinšek, M., et al. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)*, 45(3), 35.
- Deroussi, L., et al. (2008). A simple metaheuristic approach to the simultaneous scheduling of machines and automated guided vehicles. *International Journal of Production Research*, 46(8), 2143-2164.
- Du, W.-B., et al. (2015). Adequate is better: particle swarm optimization with limited-information. *Applied Mathematics and Computation*, 268, 832-838.
- Egemin Automation, I. (2016). Battery charging systems for automated guided vehicles. Retrieved from [http://www.egeminusa.com/pages/agvs/agvs\\_battery\\_charging.html](http://www.egeminusa.com/pages/agvs/agvs_battery_charging.html).
- Eiben, A. E., & Smit, S. K. (2011a). Evolutionary algorithm parameters and methods to tune them *Autonomous search* (pp. 15-36): Springer.
- Eiben, A. E., & Smit, S. K. (2011b). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1), 19-31.
- Eichfelder, G. (2008). *Adaptive scalarization methods in multiobjective optimization* (illustrated ed.): Springer Science & Business Media.
- Elsayed, S. M., et al. (2014). A new genetic algorithm for solving optimization problems. *Engineering Applications of Artificial Intelligence*, 27, 57-69.
- Ercan, F., & Li, X. (2013). Particle swarm optimization and its hybrids. *International Journal of Computer and Communication Engineering*, 2(1), 52-55.
- Eren, T., & Güner, E. (2007). Minimizing total tardiness in a scheduling problem with a learning effect. *Applied Mathematical Modelling*, 31(7), 1351-1361.
- Erol, R., et al. (2012). A multi-agent based approach to dynamic scheduling of machines and automated guided vehicles in manufacturing systems. *Applied Soft Computing*, 12(6), 1720-1732.

- Fan, S. K. S., & Zahara, E. (2007). A hybrid simplex search and particle swarm optimization for unconstrained optimization. *European Journal of Operational Research*, 181(2), 527-548.
- Farahani, R. Z., et al. (2008). Designing efficient methods for the tandem AGV network design problem using tabu search and genetic algorithm. *The International Journal of Advanced Manufacturing Technology*, 36(9-10), 996-1009.
- Fauadi, M. H. F. B. M., & Murata, T. (2010). *Makespan minimization of machines and automated guided vehicles schedule using binary particle swarm optimization*. Paper presented at the Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS), Hong Kong.
- Fazlollahtabar, H., & Saidi-Mehrabad, M. (2013). Methodologies to optimize automated guided vehicle scheduling and routing problems: a review study. *Journal of Intelligent & Robotic Systems*, 1-21.
- Fazlollahtabar, H., & Shafieian, S. H. (2014). An optimal path in an AGV-based manufacturing system with intelligent agents. *Journal for Manufacturing Science and Production*, 14(2), 87-102.
- Ficko, M., et al. (2004). Designing the layout of single-and multiple-rows flexible manufacturing system by genetic algorithms. *Journal of materials processing technology*, 157, 150-158.
- FlexSim Software Products, I. (2016). Retrieved from <https://www.flexsim.com/company/>
- Gan, Z., et al. (2013). Automated guide vehicles dynamic scheduling based on annealing genetic algorithm. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 11(5), 2508-2515.
- Gao, Y., et al. (2015). Selectively-informed particle swarm optimization. *Scientific reports*, 5.
- Gaur, D. R., et al. (2003). A 5/3-approximation algorithm for scheduling vehicles on a path with release and handling times. *Information Processing Letters*, 86(2), 87-91.
- Gelareh, S., et al. (2013). Scheduling of intelligent and autonomous vehicles under pairing/unpairing collaboration strategy in container terminals. *Transportation Research Part C: Emerging Technologies*, 33, 1-21.
- Gen, M., & Lin, L. (2012). Multiobjective genetic algorithm for scheduling problems in manufacturing systems. *Industrial Engineering and Management Systems*, 11(4), 310-330.
- Ghane-Kanafi, A., & Khorram, E. (2015). A new scalarization method for finding the efficient frontier in non-convex multi-objective problems. *Applied Mathematical Modelling*, 39(23), 7483-7498.

- Giagkiozis, I., & Fleming, P. J. (2015). Methods for multi-objective optimization: An analysis. *Information Sciences*, 293, 338-350.
- Giglio, D. (2014). *Task scheduling for multiple forklift AGVs in distribution warehouses*. Paper presented at the Emerging Technology and Factory Automation (ETFAs), IEEE, Barcelona.
- Girish, B., & Jawahar, N. (2009). *A particle swarm optimization algorithm for flexible job shop scheduling problem*. Paper presented at the 2009 IEEE International Conference on Automation Science and Engineering.
- Gnanavel Babu, A., et al. (2010). Scheduling of machines and automated guided vehicles in FMS using differential evolution. *International Journal of Production Research*, 48(16), 4683-4699.
- Gonçalves, J. F., et al. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167(1), 77-95.
- Gorcitz, R., et al. (2015). *On the scalability of constraint solving for static/off-line real-time scheduling*. Paper presented at the International Conference on Formal Modeling and Analysis of Timed Systems.
- Guo, C., & Yang, X. (2011). A programming of genetic algorithm in matlab7. 0. *Modern Applied Science*, 5(1), 230.
- Hall, N. G., et al. (2001). Operational decisions in AGV-served flowshop loops: scheduling. *Annals of Operations Research*, 107(1-4), 161-188.
- Haq, A. N., et al. (2003). Scheduling decisions in FMS using a heuristic approach. *The International Journal of Advanced Manufacturing Technology*, 22(5-6), 374-379.
- Hillston, J. (2003). Model validation and verification. *Edinburgh: University of Edinburgh*.
- Holland, J. H. (1975). *Adaptation in natural and artificial system: an introduction with application to biology, control and artificial intelligence*: The University of Michigan Press.
- Huang, D., & Zhang, G. (2013a). *Scheduling control of AGV system based on game theory*. Paper presented at the 6th International Conference on Advanced Infocomm Technology (ICAIT). Hsinchu, Taiwan.
- Huang, D., & Zhang, G. (2013b). *Scheduling control of AGV system based on game theory*. Paper presented at the Advanced Infocomm Technology (ICAIT), 2013 6th International Conference on.
- Hurink, J., & Knust, S. (2005). Tabu search algorithms for job-shop problems with a single transport robot. *European Journal of Operational Research*, 162(1), 99-111.

- Ilić, O. R. (1994). Analysis of the number of automated guided vehicles required in flexible manufacturing systems. *The International Journal of Advanced Manufacturing Technology*, 9(6), 382-389.
- Jain, S., & Foley, W. (2016). Dispatching strategies for managing uncertainties in automated manufacturing systems. *European Journal of Operational Research*, 248(1), 328-341.
- Jamrus, T., et al. (2013). *Hybrid particle swarm optimization with genetic operators and cauchy distribution for flexible job-shop scheduling problem*. Paper presented at the 14th Asia Pacific Industrial Engineering and Management Systems, Cebu, Philippines.
- Jans, R., & Degraeve, Z. (2007). Meta-heuristics for dynamic lot sizing: A review and comparison of solution approaches. *European Journal of Operational Research*, 177(3), 1855-1875.
- Jerald, J., et al. (2005). Scheduling optimisation of flexible manufacturing systems using particle swarm optimisation algorithm. *The International Journal of Advanced Manufacturing Technology*, 25(9-10), 964-971.
- Jerald, J., et al. (2006). Simultaneous scheduling of parts and automated guided vehicles in an FMS environment using adaptive genetic algorithm. *The International Journal of Advanced Manufacturing Technology*, 29(5-6), 584-589.
- Jiang, F., et al. (2017). A new binary hybrid particle swarm optimization with wavelet mutation. *Knowledge-Based Systems*, 130, 90-101.
- Joshi, G. (2014). Review of genetic algorithm: an optimization technique. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(4), 802-805.
- Kao, Y. T., & Zahara, E. (2008). A hybrid genetic algorithm and particle swarm optimization for multimodal functions. *Applied Soft Computing*, 8(2), 849-857.
- Kaplanoğlu, V., et al. (2014). *A multi-agent based approach to dynamic scheduling of machines and automated guided vehicles (AGV) in manufacturing systems by considering AGV breakdowns*. Paper presented at the 4th IAJC/ISAM Joint International Conference on engineering and related technologies, Orlando, Florida.
- Karafotias, G., et al. (2015). Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Trans. Evolutionary Computation*, 19(2), 167-187.
- Karsak, E. E., & Kuzgunkaya, O. (2002). A fuzzy multiple objective programming approach for the selection of a flexible manufacturing system. *International Journal of Production Economics*, 79(2), 101-111.
- Karthikeyan, S., et al. (2015). A hybrid discrete firefly algorithm for solving multi-objective flexible job shop scheduling problems. *International Journal of Bio-Inspired Computation*, 7(6), 386-401.

- Kato, F., & Shin, S. (2010). *Multistep optimal scheduling of automated guided vehicles in a semiconductor fabrication*. Paper presented at the Proceedings of SICE Annual Conference 2010.
- Kaveh, A., & Malakouti Rad, S. (2010). Hybrid genetic algorithm and particle swarm optimization for the force method-based simultaneous analysis and design. *Iranian Journal of Science and Technology, Transaction B: Engineering*, 34(B1), 15-34.
- Kawakami, T., & Takata, S. (2012). Battery life cycle management for automatic guided vehicle systems. *Design for Innovative Value Towards a Sustainable Society* (pp. 403-408): Springer.
- Keesman, K. J. (2011). Model validation techniques *System identification: An introduction* (pp. 225-247). London: Springer London.
- Kelton, W. D., et al. (2004). *Simulation with Arena*, 3rd: New York: McGraw-Hill.
- Kennedy, J., & Eberhart, R. C. (1995). *Particle swarm optimization*. Paper presented at the IEEE international conference on neural networks, Piscataway, New Jersey.
- Kennedy, J., et al. (2001). *Swarm intelligence*. US: Morgan Kaufmann.
- Kessentini, S., & Barchiesi, D. (2015). Particle swarm optimization with adaptive inertia weight. *International Journal of Machine Learning and Computing*, 5(5), 368.
- Khanmohammadi, S., et al. (2010). *Multi AGV hybrid path planning using fuzzy inference systems*. Paper presented at the The 2nd International Conference on Computer and Automation Engineering (ICCAE).
- Kim, D., et al. (2007). Hybrid genetic: Particle swarm optimization algorithm. In C. G. Ajith Abraham, Hisao Ishibuchi (Ed.), *Hybrid Evolutionary Algorithms* (Vol. 75, pp. 147-170): Springer.
- Kruse, R., et al. (2013). *Synergies of soft computing and statistics for intelligent data analysis*: Springer.
- Kumar, M. S., et al. (2011). Simultaneous scheduling of machines and vehicles in an FMS environment with alternative routing. *The International Journal of Advanced Manufacturing Technology*, 53(1-4), 339-351.
- Kuo, H., et al. (2009). An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model. *Expert Systems with Applications*, 36(3), 7027-7032.
- Kuo, R., & Lin, L. (2010). Application of a hybrid of genetic algorithm and particle swarm optimization algorithm for order clustering. *Decision Support Systems*, 49(4), 451-462.
- Lasi, H., et al. (2014). Industry 4.0. *Business & Information Systems Engineering*, 6(4), 239.



- Le-Anh, T. (2005). *Intelligent control of vehicle-based internal transport systems*: Erasmus Research Institute of Management (ERIM).
- Le-Anh, T., & De Koster, M. (2006). A review of design and control of automated guided vehicle systems. *European Journal of Operational Research*, 171(1), 1-23.
- Li, K., et al. (2014). An agent-based intelligent algorithm for uniform machine scheduling to minimize total completion time. *Applied Soft Computing*, 25, 277-284.
- Li, W., et al. (2015). *Planning and scheduling for maritime container yards: supporting and facilitating the global supply network*: Springer.
- Li, X., & Yao, X. (2012). Cooperatively coevolving particle swarms for large scale optimization. *Evolutionary Computation, IEEE Transactions on*, 16(2), 210-224.
- Liang, Y., et al. (2012). *A hybrid evolutionary algorithm for FMS optimization with AGV dispatching*. Paper presented at the Computers and Industrial Engineering 42, Cape Town, South Africa.
- Lin, J. J., et al. (2014). Optimal AGV configuration by simulation of flow shop scheduling in an assembly plant. In H. S. Peilong Xu, Yiqian Wang and Pin Wang (Ed.), *Advanced Materials Research* (Vol. 926-930, pp. 3132-3136): Trans Tech Publ.
- Liou, C. D., et al. (2013). A new encoding scheme-based hybrid algorithm for minimising two-machine flow-shop group scheduling problem. *International Journal of Systems Science*, 44(1), 77-93.
- Liu, L. L., et al. (2015). A hybrid PSO-GA algorithm for job shop scheduling in machine tool production. *International Journal of Production Research*, 53(19), 1-27.
- Lobo, F., et al. (2007). *Parameter setting in evolutionary algorithms* (Vol. 54): Springer Science & Business Media.
- Malhotra, V., et al. (2010). Excellent techniques of manufacturing systems: RMS and FMS. *International Journal of Engineering Science and Technology*, 2(3), 137-142.
- Mallikarjuna, V., Md.Jaffar, Jadesh, Hanumantaraya. (2014). A review on parallel scheduling of machines and AGV'S in an FMS environment. *International Journal of Computational Engineering Research (IJCER)*, 4(5), 2250-3005.
- Marler, R. T., & Arora, J. S. (2010). The weighted sum method for multi-objective optimization: new insights. *Structural and multidisciplinary optimization*, 41(6), 853-862.
- Mateo, J. R. S. C. (2012). Weighted sum method and weighted product method *Multi Criteria Analysis in the Renewable Energy Industry* (pp. 19-22): Springer.
- Mehta, M. (2012). Hybrid genetic algorithm with PSO effect for combinatorial optimization problems. *International Journal of Advanced Computer Research*, 2(4), 300-305.

- Mezura-Montes, E., & Coello Coello, C. (2004). *An improved diversity mechanism for solving constrained optimization problems using a multimembered evolution strategy*. Paper presented at the Genetic and Evolutionary Computation–GECCO 2004.
- Morandin, O., et al. (2011). *Adaptive genetic fuzzy, predictive and multiobjective approach for AGVs dispatching*. Paper presented at the IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society.
- Morandin, O., et al. (2010). *A strategy of production scheduling with the fitness function of genetic algorithm using Timed Petri net and considering AGV and the input buffer*. Paper presented at the IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society.
- Mousavi, M., et al. (2017). Multi-objective AGV scheduling in an FMS using a hybrid of genetic algorithm and particle swarm optimization. *Plos one*, 12(3), e0169817.
- Naderi, B., et al. (2010). Electromagnetism-like mechanism and simulated annealing algorithms for flowshop scheduling problems minimizing the total weighted tardiness and makespan. *Knowledge-Based Systems*, 23(2), 77-85.
- Nageswararao, M., et al. (2014). Simultaneous scheduling of machines and AGVs in flexible manufacturing system with mean tardiness criterion by using HGVA. *INROADS-An International Journal of Jaipur National University*, 3(1s), 62-68.
- Nanvala, H. (2011). Use of Genetic algorithm based approaches in scheduling of FMS: A Review. *International Journal of Engineering Science and Technology*, 3(3), 1936-1942.
- Nearchou, A. C. (2004). The effect of various operators on the genetic search for large scheduling problems. *International Journal of Production Economics*, 88(2), 191-203.
- Novas, J. M., & Henning, G. P. (2014). Integrated scheduling of resource-constrained flexible manufacturing systems using constraint programming. *Expert Systems with Applications*, 41(5), 2286-2299.
- Oliveira, M. M., et al. (2012). *NICD battery discharging estimation system for AGVs working in intelligent warehouses based on EKF*. Paper presented at the ABCM Symposium Series in Mechatronics.
- Oliveira, M. M., et al. (2011). *Battery state estimation for applications in intelligent warehouses*. Paper presented at the IEEE International Conference on Robotics and Automation (ICRA), Shanghai.
- Oyetunji, E. (2012, 3-6 July). *Minimizing the sum of makespan and maximum tardiness on single machine with release dates*. Paper presented at the International Conference on Industrial Engineering and Operations Management, Istanbul, Turkey.

- Özğüven, C., et al. (2010). Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 34(6), 1539-1548.
- Palonen, M., et al. (2009). *A genetic algorithm for optimization of building envelope and HVAC system parameters*. Paper presented at the Proc. Of the 11th IBPSA Conference, Glasgow, Scotland.
- Pan, X. Y., et al. (2013). A case study of AGV scheduling for production material handling. *Applied Mechanics and Materials*, 411, 2351-2354.
- Pandey, R., et al. (2016). Performance evaluation of flexible manufacturing system (FMS) in manufacturing industries. *Imperial Journal of Interdisciplinary Research*, 2(3), 176-180.
- Pezzella, F., et al. (2008). A genetic algorithm for the flexible job-shop scheduling problem. *Computers & operations research*, 35(10), 3202-3212.
- Pinedo, M. (2012). *Scheduling*: Springer.
- Pinedo, M. L. (2016). *Scheduling: theory, algorithms, and systems*: Springer.
- Pongchairerks, P. (2009). Particle swarm optimization algorithm applied to scheduling problems. *ScienceAsia*, 35(1), 89-94.
- Premalatha, K., & Natarajan, A. (2009). Hybrid PSO and GA for global maximization. *International Journal of Open Problems in Computational Mathematics*, 2(4), 597-608.
- Premalatha, K., & Natarajan, A. (2010). Hybrid PSO and GA models for Document Clustering. *International Journal of Advances in Soft Computing and Its Applications*, 2(3), 302-320.
- Qiu, L., et al. (2002). Scheduling and routing algorithms for AGVs: a survey. *International Journal of Production Research*, 40(3), 745-760.
- Rashmi, M., & Bansal, S. (2014). Task scheduling of automated guided vehicle in flexible manufacturing system using ant colony optimization. *International Journal of Latest Trends in Engineering and Technology (IJLTET)*, 4(1), 177-181.
- Reddy, B., & Rao, C. (2006). A hybrid multi-objective GA for simultaneous scheduling of machines and AGVs in FMS. *The International Journal of Advanced Manufacturing Technology*, 31(5-6), 602-613.
- Reddy, B., & Rao, C. (2011). Flexible manufacturing systems modelling and performance evaluation using AutoMod. *International Journal of Simulation Modelling*, 10(2), 78-90.
- Ren, N. F., et al. (2013). AGV scheduling optimizing research of collaborative manufacturing system based on improved genetic algorithm. *Applied Mechanics and Materials*, 300, 55-61.

- Rüßmann, M., et al. (2015). Industry 4.0: The future of productivity and growth in manufacturing industries. *Boston Consulting Group*, 14.
- Sabuncuoğlu, I., & Bayız, M. (2000). Analysis of reactive scheduling problems in a job shop environment. *European Journal of Operational Research*, 126(3), 567-586.
- Saidi-Mehrabad, M., et al. (2015). An ant colony algorithm (ACA) for solving the new integrated model of job shop scheduling and conflict-free routing of AGVs. *Computers & industrial engineering*, 86, 2-13.
- Salehipour, A., et al. (2011). Locating workstations in tandem automated guided vehicle systems. *The International Journal of Advanced Manufacturing Technology*, 52(1-4), 321-328.
- Samuel, G. G., & Rajan, C. C. A. (2015). Hybrid: particle swarm optimization–genetic algorithm and particle swarm optimization–shuffled frog leaping algorithm for long-term generator maintenance scheduling. *International Journal of Electrical Power & Energy Systems*, 65, 432-442.
- Sawada, K., et al. (2013). Optimal scheduling of automatic guided vehicle system via state space realization. *Journal ref: International Journal of Automation Technology*, 7(5), 571-580.
- Sha, D., & Lin, H.-H. (2010). A multi-objective PSO for job-shop scheduling problems. *Expert Systems with Applications*, 37(2), 1065-1070.
- Shabtay, D., et al. (2013). A survey on offline scheduling with rejection. *Journal of Scheduling*, 16(1), 3-28.
- Shi, Y., & Eberhart, R. C. (1999). *Empirical study of particle swarm optimization*. Paper presented at the Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on.
- Shirazi, B., et al. (2010). A six sigma based multi-objective optimization for machine grouping control in flexible cellular manufacturing systems with guide-path flexibility. *Advances in Engineering Software*, 41(6), 865-873.
- Shukla, A., et al. (2015). *Comparative review of selection techniques in genetic algorithm*. Paper presented at the Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), 2015 International Conference on.
- Silberholz, J., & Golden, B. (2010). Comparison of metaheuristics *Handbook of metaheuristics* (pp. 625-640): Springer.
- Sinha, A., et al. (2014). *A bilevel optimization approach to automated parameter tuning*. Paper presented at the Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation.
- Sinriech, D., & Kotlarski, J. (2002). A dynamic scheduling algorithm for a multiple-load multiple-carrier system. *International Journal of Production Research*, 40(5), 1065-1080.

- Soleimani, H., & Kannan, G. (2015). A hybrid particle swarm optimization and genetic algorithm for closed-loop supply chain network design in large-scale networks. *Applied Mathematical Modelling*, 39(14), 3990-4012.
- Song, M. L. (2014). *A study of single-objective particle swarm optimization and multi-objective particle swarm optimization*. Paper presented at the Applied Mechanics and Materials.
- Song, X. (2010). Hybrid particle swarm algorithm for job shop scheduling problems. In T. Aized (Ed.), *Future Manufacturing Systems* (pp. 235-267): INTECH Open Access Publisher.
- Spears, W. M., & Anand, V. (1991). *A study of crossover operators in genetic programming*: Springer.
- Suzuki, T., et al. (2014). *Influence of the number of AGVs on products conveyance efficiency in AGV transportation system based on knowledge of Taxis*. Paper presented at the ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, New York, USA.
- Tang, J., et al. (2010). A hybrid PSO/GA algorithm for job shop scheduling problem *Advances in Swarm Intelligence* (pp. 566-573): Springer Berlin Heidelberg.
- Tasgetiren, M. F., et al. (2006). Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem. *International Journal of Production Research*, 44(22), 4737-4754.
- Tasgetiren, M. F., et al. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, 177(3), 1930-1947.
- Tasgetiren, M. F., et al. (2004). *Particle swarm optimization algorithm for single machine total weighted tardiness problem*. Paper presented at the Evolutionary Computation. CEC2004.
- Tavakkoli-Moghaddam, R., et al. (2008). Partitioning machines in tandem AGV systems based on “balanced flow strategy” by simulated annealing. *The International Journal of Advanced Manufacturing Technology*, 38(3-4), 355-366.
- Thakur, M., et al. (2014). A modified real coded genetic algorithm for constrained optimization. *Applied Mathematics and Computation*, 235, 292-317.
- The MathWorks, I. (2016). Retrieved from <https://www.mathworks.com/products/matlab/features.html>.
- Tompkins, J. A., et al. (2010). *Facilities planning*: John Wiley & Sons.
- Tseng, M. C. (2004). Strategic choice of flexible manufacturing technologies. *International Journal of Production Economics*, 91(3), 223-227.

- Udhayakumar, P., & Kumanan, S. (2010). Task scheduling of AGV in FMS using non-traditional optimization techniques. *International Journal of Simulation Modelling*, 9(1), 28-39.
- Udhayakumar, P., & Kumanan, S. (2012). Integrated scheduling of flexible manufacturing system using evolutionary algorithms. *The International Journal of Advanced Manufacturing Technology*, 61(5-8), 621-635.
- Ullrich, C. A. (2013). Integrated machine scheduling and vehicle routing with time windows. *European Journal of Operational Research*, 227(1), 152-165.
- Ulusoy, G., et al. (1997). A genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles. *Computers & operations research*, 24(4), 335-351.
- Umar, U. A., et al. (2015). Hybrid multiobjective genetic algorithms for integrated dynamic scheduling and routing of jobs and automated-guided vehicle (AGV) in flexible manufacturing systems (FMS) environment. *The International Journal of Advanced Manufacturing Technology*, 81(9-12), 2123-2141.
- Valdez, F., et al. (2008). *A new evolutionary method with a hybrid approach combining particle swarm optimization and genetic algorithms using fuzzy logic for decision making*. Paper presented at the Evolutionary Computation. CEC 2008.(IEEE World Congress on Computational Intelligence).
- Vasava, A. S. (2014). Scheduling of automated guided vehicle in different flexible manufacturing system environment. *International Journal of Innovative Research in Advanced Engineering (IJIRAE)*, 1(8), 262-267.
- Veček, N., et al. (2014). A chess rating system for evolutionary algorithms: a new method for the comparison and ranking of evolutionary algorithms. *Information Sciences*, 277, 656-679.
- Veček, N., et al. (2016). Parameter tuning with chess rating system (CRS-Tuning) for metaheuristic algorithms. *Information Sciences*, 372, 446-469.
- Veeravalli, B., et al. (2002). Design and analysis of optimal material distribution policies in flexible manufacturing systems using a single AGV. *International Journal of Production Research*, 40(12), 2937-2954.
- Ventura, J. A., et al. (2015). Finding optimal dwell points for automated guided vehicles in general guide-path layouts. *International Journal of Production Economics*, 170, 850-861.
- Vis, I. F. (2006). Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170(3), 677-709.
- Vivaldini, K., et al. (2013). *Battery charge state estimate for a robotic forklift routing system*. Paper presented at the Industrial Technology (ICIT), 2013 IEEE International Conference on.

- Wall, M. B. (1996). *A genetic algorithm for resource-constrained scheduling*. (Doctoral dissertation), Massachusetts Institute of Technology.
- Wang, H. F., & Chan, C. H. (2014). Multi-objective optimisation of automated guided dispatching and vehicle routing system. *International Journal of Modelling in Operations Management*, 4(1), 35-52.
- Wang, J., et al. (2016). Hybrid forecasting model-based data mining and genetic algorithm-adaptive particle swarm optimisation: a case study of wind speed time series. *IET Renewable Power Generation*, 10(3), 287-298.
- Wang, J. B., et al. (2014). Simulating an AGV scheduling in job workshop for optimal configuration. In H. S. Peilong Xu, Yiqian Wang and Pin Wang (Ed.), *Advanced Materials Research* (Vol. 926-930, pp. 1562-1565): Trans Tech Publ.
- Wang, L., & Si, G. (2010). *Optimal location management in mobile computing with hybrid genetic algorithm and particle swarm optimization (GA-PSO)*. Paper presented at the Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on.
- Wang, S., et al. (2016). Implementing smart factory of industrie 4.0: an outlook. *International Journal of Distributed Sensor Networks*.
- Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1), 67-82.
- Wu, C.-H., et al. (2010). Chaotic hybrid algorithm and its application in circle detection *Applications of Evolutionary Computation* (pp. 302-311): Springer.
- Xia, W., & Wu, Z. (2005). An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & industrial engineering*, 48(2), 409-425.
- Xia, W., & Wu, Z. (2006). A hybrid particle swarm optimization approach for the job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 29(3-4), 360-366.
- Yahyaeei, M., et al. (2010). Controlling the navigation of automatic guided vehicle (AGV) using integrated fuzzy logic controller with programmable logic controller (IFLPLC)—stage 1. *The International Journal of Advanced Manufacturing Technology*, 47(5-8), 795-807.
- Yu, S., et al. (2015). A hybrid self-adaptive particle swarm optimization–genetic algorithm–radial basis function model for annual electricity demand prediction. *Energy Conversion and Management*, 91, 176-185.
- Yu, X., & Gen, M. (2010a). Advanced evolutionary algorithms. *Introduction to Evolutionary Algorithms*, 39-132.
- Yu, X., & Gen, M. (2010b). Simple evolutionary algorithms. *Introduction to Evolutionary Algorithms*, 11-38.

- Zeng, C., et al. (2014). Scheduling of no buffer job shop cells with blocking constraints and automated guided vehicles. *Applied Soft Computing*, 24, 1033-1046.
- Zhang, G., et al. (2011). An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 38(4), 3563-3573.
- Zhang, G., et al. (2009). An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Computers & industrial engineering*, 56(4), 1309-1318.
- Zhao, W., et al. (2013). *The multi-task scheduling and controlling simulation method of the AGVS*. Paper presented at the Proceedings of the World Congress on Engineering and Computer Science, San Francisco, USA.
- Zheng, K., et al. (2013). Distributed control of multi-AGV system based on regional control model. *Production Engineering*, 7(4), 433-441.
- Zheng, Y., et al. (2014). A tabu search algorithm for simultaneous machine/AGV scheduling problem. *International Journal of Production Research*, 52(19), 5748-5763.
- Zini, H., & ElBernoussi, S. (2015). A discrete particle swarm optimization with combined priority dispatching rules for hybrid flow shop scheduling problem. *Applied Mathematical Sciences*, 9(24), 1175-1187.



## LIST OF PUBLICATIONS AND PAPERS PRESENTED

### JOURNALS

1. **Maryam Mousavi**, Hwa Jen Yap, Siti Nurmaya Musa, Farzad Tahriri, Siti Zawiah Md Dawal. (2017). Multi-Objective AGV Scheduling in an FMS Using a Hybrid of Genetic Algorithm and Particle Swarm Optimization. *PloS one*, 12(3), e0169817. Doi: <https://doi.org/10.1371/journal.pone.0169817>.
2. **Maryam Mousavi**, Hwa Jen Yap, Siti Nurmaya Musa, Siti Zawiah Md Dawal. (2017). A Fuzzy Hybrid GA-PSO Algorithm for Multi-Objective AGV Scheduling in an FMS. *International Journal of Simulation Modelling (IJSIMM)*, 16(1), pp 58-71.
3. **Maryam Mousavi**, Hwa Jen Yap, Siti Nurmaya Musa, Siti Zawiah Md Dawal. Automated Guided Vehicle Systems: Indoor and Outdoor Dispatching, Scheduling, and Routing. *International Journal of Industrial Engineering: Theory, Applications and Practice*. (Under review from 25/6/2015).
4. **Maryam Mousavi**, Yap Hwa Jen, Siti Nurmaya, (2013). A Review on Cybersickness and Usability in Virtual Environments. *Advanced Engineering Forum*, Vol. 10, pp. 34-39. doi: 10.4028/www.scientific.net/AEF.10.34.
5. Farzad Tahriri, **Maryam Mousavi**, Hwa Jen Yap, Siti Zawiah Md Dawal, and Zahari Taha. (2015). Optimizing the Robot Arm Movement Time Using Virtual Reality Robotic Teaching System. *International Journal of Simulation Modelling*, 14(1), pp 28-38.
6. Siti Zawiah Md Dawal, Farzad Tahriri, Yap Hwa Jen, Keith Case, Nguyen Huu Tho, Aliq Zuhdi and, **Maryam Mousavi**, Atefeh Amindoust, Novita Sakundarini, (2015), Empirical Evidence of AMT Practices and Environmental Initiatives in Malaysian Automotive SMEs, *International Journal of Precision Engineering and Manufacturing*. Volume 16, Issue 6, pp 1195-1203.

### CONFERENCES

1. **Maryam Mousavi**, Yap Hwa Jen, Siti Nurmaya, Farzad Tahriri, (2014) Application of Automated Guided Vehicle system in industry, International Conference on Engineering and Applied Science, Tokyo, Japan.

## APPENDIX: MODEL AND ALGORITHMS PROGRAMMING

### GA-function

```
function [output] = GA_func(g, problem)
%function [gaDat, output] = GA_func(g, problem)

output.History.Y.Makespan = [inf];
output.History.Y.NAGV = [inf];
output.History.Y.Eval = [inf];
output.History.X = ones(1, g.D) .* inf;
output.History.chrom = ones(1, g.D) .* inf;

g.MAXGEN = g.MaxGen-1;
D = ones(1, g.D);
g.FieldD = [D.* g.LB; D.*g.UB];
g.NIND = g.N;

gaDat=g;

% If the parameter doesn't exist in the data structure it is created
with the default value
if ~isfield(gaDat, 'NVAR')
    gaDat.NVAR=size(gaDat.FieldD,2);
end
if ~isfield(gaDat, 'MAXGEN')
    gaDat.MAXGEN=gaDat.NVAR*20+10;
end
if ~isfield(gaDat, 'NIND')
    gaDat.NIND=gaDat.NVAR*50;
end
if ~isfield(gaDat, 'alfa')
    gaDat.alfa=0;
end
if ~isfield(gaDat, 'Pc')
    gaDat.Pc=0.9;
end
if ~isfield(gaDat, 'Pm')
    gaDat.Pm=0.1;
end
if ~isfield(gaDat, 'indini')
    gaDat.indini=[];
end

% Internal parameters
gaDat.Chrom=[];
gaDat.ObjV=[];
gaDat.xmin=[];
gaDat.fxmin=inf;
gaDat.xmingen=[];
gaDat.fxmingen=[];
gaDat.rf=(1:gaDat.NIND)';
gaDat.gen=0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Main loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Generation counter
gen=0;
```

```

% Initial population -----
gaDat.Chrom=crtrp(gaDat.NIND,gaDat.FieldD); % Real codification
% Individuals of gaDat.indini are randomly added in the initial
population
if not isempty(gaDat.indini)
    nind0=size(gaDat.indini,1);
    posicion0=ceil(rand(1,nind0)*gaDat.NIND);
    gaDat.Chrom(posicion0,:)=gaDat.indini;
end

while (gaDat.gen<gaDat.MAXGEN),
    gaDat.gen=gen;
    [gaDat, output, problem]=gaevolucion(gaDat, output, problem);
    % Increase generation counter -----
    gaDat.xmingen(gen+1,:)=gaDat.xmin;
    gaDat.fxmingen(gen+1,:)=gaDat.fxmin;
    gen=gen+1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End main loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Present final results

%% Subfunction -----

%% -----
function chrom=crtrp(Nind,FieldDR)
% A random real value matrix is created coerced by upper and
% lower bounds

Nvar = size(FieldDR,2);
aux = rand(Nind,Nvar);
m=[-1 1]*FieldDR;
ublb=ones(Nind,1)*m;
lb=ones(Nind,1)*FieldDR(1,:);
chrom=ublb.*aux+lb;

%% -----
function [gaDat, output, problem] = gaevolucion(gaDat, output,
problem)
% One generation -----
output.chrom=gaDat.Chrom;
nind=size(output.chrom,1);
ObjV=inf(nind,1);
[ output ] = SPV( output.chrom, problem.Job, output );
for i=1:nind
    [output]=Fitness(i, problem, output);
    ObjV(i) = output.Y.Eval(i);
end
output.History.Y.Makespan = [output.History.Y.Makespan ;
output.Y.Makespan];
output.History.Y.NAGV = [output.History.Y.NAGV ; output.Y.NAGV];
output.History.Y.Eval = [output.History.Y.Eval ; output.Y.Eval];
output.History.X = [output.History.X ; output.X];
output.History.chrom = [output.History.chrom ; output.chrom];

gaDat.ObjV=ObjV;

% Best individual of the generation -----
[v,p]=min(gaDat.ObjV);
if v<=gaDat.fxmin
    gaDat.xmin=output.chrom(p,:);

```

```

    gaDat.fxmin=v;
end
% Next generation
% RANKING -----
FitnV = ranking(gaDat.ObjV,gaDat.rf);
% SELECTION -----
% Stochastic Universal Sampling (SUS).
SelCh = select('sus',output.chrom,FitnV,1);
% CROSSOVER -----
% Uniform crossover.
SelCh = lxov(SelCh,gaDat.Pc,gaDat.alfa);
% MUTATION -----
output.chrom = mutbga(SelCh,gaDat.FieldD,[gaDat.Pm 1]); % Codificación
Real.
% Reinsert the best individual -----
output.chrom(round(gaDat.NIND/2),:) = gaDat.xmin;
gaDat.Chrom=output.chrom;
% Optional additional task required by user

%% -----
function FitV=ranking(ObjV,RFun)
% Ranking function
if nargin==1
    error('Ranking function needs two parameters');
end

if ~(length(ObjV)==length(RFun))
    error('RFun have to be of the same size than ObjV.');
```

```

end

[~,pos]=sort(ObjV);
FitV(pos)=flipud(RFun);
FitV=FitV';

%% -----
function [SelCh]=select(SEL_F, chrom, FitnV, GGAP)
% Selection Function
if (nargin==3) % No overlap -----
    if strcmp(SEL_F,'rws')
        % Roulette wheel selection method
        indices=rws(FitnV,length(FitnV));
        SelCh=chrom(indices,:);
    elseif strcmp(SEL_F,'sus')
        % Stochastic universal sampling selection
        indices=sus(FitnV,length(FitnV));
        SelCh=chrom(indices,:);
    else
        error('Incorrect selection method');
    end
elseif (nargin==4) % With overlap -----
    % Indexes of new individuals
    if strcmp(SEL_F,'rws')
        indices=rws(FitnV,round(length(FitnV)*GGAP));
    elseif strcmp(SEL_F,'sus')
        indices=sus2(FitnV,round(length(FitnV)*GGAP));
    else
        error('Incorrect selection method');
    end

    if (GGAP<1) % there is overlap
        % Members of the population to overlap
        oldpos=(1:length(FitnV))';
        for k=1:length(FitnV)
```

```

        pos=round(rand*length(FitnV)+0.5);
        % exchange indexes
        oldpos([pos k])=oldpos([k pos]);
    end
    oldpos=oldpos(1:round(length(FitnV)*GGAP));
    SelCh=chrom;
    SelCh(oldpos,:)=chrom(indices,:);
else % more childs than parents
    SelCh=chrom(indices,:);
end
else
    error('Incorrect number of paramenterers');
end

% Disorder the population.
[~,indi]=sort(rand(length(FitnV),1));
SelCh=SelCh(indi,:);

%% -----
function NewChrom =lxov(OldChrom, XOVR, alpha)
% Linear crossover
% Produce a~ new population by linear crossover and XOVR crossover
probability
%   NewChroms =lxov(OldChrom, XOVR, alpha, FieldDR)
%
% Linear recombination.
% Parameters 'beta1' and 'beta2' are randomly obtained inside [-alpha,
1+alpha]
% interval
%   Child1 = beta1*Parent1+(1-beta1)*Parent2
%   Child2 = beta2*Parent1+(1-beta2)*Parent2

if nargin==1
    XOVR = 0.7;
    alpha = 0;
elseif nargin==2
    alpha = 0;
end

n = size(OldChrom,1); % Number of individuals and chromosome length
npares = floor(n/2); % Number of pairs
cruzar = rand(npares,1)<= XOVR; % Pairs to crossover
NewChrom=OldChrom;

for i=1:npares
    pin = (i-1)*2+1;
    if ~(cruzar(i)==0)
        betas=rand(2,1)*(1+2*alpha)-(0.5+alpha);
        A=[betas(1) 1-betas(1); 1-betas(2) betas(2)];
        NewChrom(pin:pin+1,:)=A*OldChrom(pin:pin+1,:);
    end
end

% Coerce points outside search space
% aux = ones(n,1);
% auxf1=aux*FieldDR(1,:);
% auxf2=aux*FieldDR(2,:);
% NewChrom =
(NewChrom>auxf2).*auxf2+(NewChrom<auxf1).*auxf1+(NewChrom<=auxf2 &
NewChrom>=auxf1).*NewChrom;

%% -----

```

```

function NewChrom=mutbga(OldChrom,FieldDR,MutOpt)
% Mutation function
% Real coded mutation.
% Mutation is produced adding a low random value
% OldChrom: Initial population.
% FieldChrom: Upper and lower bounds.
% MutOpt: mutation options,
%     MutOpt(1)=mutation probability (0 to 1).
%     MutOpt(2)=compression of the mutation value (0 to 1).
%     default MutOpt(1)=1/Nvar y MutOpt(2)=1

if (nargin==3)
    pm=MutOpt(1);
    shr=MutOpt(2);
elseif (nargin==2)
    pm=1/size(FieldDR,2);
    shr=1;
else
    error('Incorrect number of parameters');
end

Nind=size(OldChrom,1);
m1=0.5-(1-pm)*0.5;
m2=0.5+(1-pm)*0.5;
aux=rand(size(OldChrom));
MutMx=(aux>m2)-(aux<m1);
range=[-1 1]*FieldDR*0.5*shr;
range=ones(Nind,1)*range;
index=find(MutMx);
m=20;
alpha=rand(m,length(index))<(1/m);
xx=2.^(0:-1:(1-m));
aux2=xx*alpha;
delta=zeros(size(MutMx));
delta(index)=aux2;
NewChrom=OldChrom+(MutMx.*range.*delta);

% Coerce points outside bounds
aux = ones(Nind,1);
auxf1=aux*FieldDR(1,:);
auxf2=aux*FieldDR(2,:);
NewChrom =
(NewChrom>auxf2).*auxf2+(NewChrom<auxf1).*auxf1+(NewChrom<=auxf2 &
NewChrom>=auxf1).*NewChrom;

%% -----
function NewChrIx=sus2(FitnV, Nsel)
suma=sum(FitnV);
% Position of the roulette pointers
j=0;
sumfit=0;
paso=suma/Nsel; % distance between pointers
flecha=rand*paso; % offset of the first pointer
NewChrIx(Nsel,1)=0;
for i=1:Nsel
    sumfit=sumfit+FitnV(i);
    while (sumfit>=flecha)
        j=j+1;
        NewChrIx(j)=i;
        flecha=flecha+paso;
    end
end
end
%% -----

```

## PSO\_function

```
function [output]= PSO_func(Parameter, problem)
%function [gbest,gbestval,fitcount,Fx, output]= PSO_func(Parameter,
problem)

Dimension = Parameter.D;
Particle_Number = Parameter.N;
Max_Gen = Parameter.MaxGen;
VRmin = Parameter.LB;
VRmax = Parameter.UB;

%----- output initialization -----
output.History.Y.Makespan = [inf];
output.History.Y.NAGV = [inf];
output.History.Y.Eval = [inf];
output.History.X = ones(1, Dimension) .* inf;
output.History.pos = ones(1, Dimension) .* inf;

%-----
rand('state',sum(100*clock));
me=Max_Gen;
ps=Particle_Number;
D=Dimension;
iwtmax = Parameter.InertiaWeight(2) - Parameter.InertiaWeight(1);
cc=Parameter.C; %acceleration constants
iwt=rand(me,1).*(Parameter.InertiaWeight(1) + iwtmax;%0.9-
(1:me).*(0.5./me));%

if length(VRmin)==1
    VRmin= repmat(VRmin,1,D);
    VRmax= repmat(VRmax,1,D);
end
mv=0.5*(VRmax-VRmin);
VRmin= repmat(VRmin,ps,1);
VRmax= repmat(VRmax,ps,1);
Vmin= repmat(-mv,ps,1);
Vmax=-Vmin;
output.pos=VRmin+(VRmax-VRmin).*rand(ps,D);

[ output ] = SPV( output.pos, problem.Job, output );

for ieval=1:size(output.pos,1)
    [output]=Fitness(ieval, problem, output);
    %Sphere(output.pos(ieval,:), size(output.pos,2));
    e(ieval) = output.Y.Eval(ieval);
end
Fx=e;

output.History.Y.Makespan = [output.History.Y.Makespan ;
output.Y.Makespan];
output.History.Y.NAGV = [output.History.Y.NAGV ; output.Y.NAGV];
output.History.Y.Eval = [output.History.Y.Eval ; output.Y.Eval];
output.History.X = [output.History.X ; output.X];
output.History.pos = [output.History.pos ; output.pos];

fitcount=ps;
vel=Vmin+2.*Vmax.*rand(ps,D);%initialize the velocity of the particles
pbest=output.pos;
pbestval=e; %initialize the pbest and the pbest's fitness value
[gbestval,gbestid]=min(pbestval);
gbest=pbest(gbestid,:);%initialize the gbest and the gbest's fitness
value
```

```

gbestrep= repmat(gbest,ps,1);

%----- Iteration -----
for i=2:me

    Prpbest = cc(1).*rand(ps,D);
    Prgbest = cc(2).*rand(ps,D);

    aa=Prpbest.*(pbest-output.pos) + Prgbest.*(gbestrep-output.pos);

    vel=iwt(i).*vel+aa;%
    vel=(vel>Vmax).*Vmax+(vel<=Vmax).*vel;
    vel=(vel<Vmin).*Vmin+(vel>=Vmin).*vel;
    output.pos=output.pos+vel;

output.pos=( (output.pos>=VRmin) & (output.pos<=VRmax) ).*output.pos...
    +(output.pos<VRmin).*(VRmin + 1.*(VRmax-VRmin).*rand(ps,D))...
    +(output.pos>VRmax).*(VRmax - 1.*(VRmax-VRmin).*rand(ps,D));

    [ output ] = SPV( output.pos, problem.Job, output);
    for ieval=1:size(output.pos,1)
        [output]=Fitness(ieval, problem, output);
    %Sphere(output.pos(ieval,:), size(output.pos,2));
        e(ieval) = output.Y.Eval(ieval);
    end
    Fx=[Fx;e];

    output.History.Y.Makespan = [output.History.Y.Makespan ;
output.Y.Makespan];
    output.History.Y.NAGV = [output.History.Y.NAGV ; output.Y.NAGV];
    output.History.Y.Eval = [output.History.Y.Eval ; output.Y.Eval];
    output.History.X = [output.History.X ; output.X];
    output.History.pos = [output.History.pos ; output.pos];

    fitcount=fitcount+ps;
    tmp=(pbestval<e);
    temp=repmat(tmp',1,D);
    pbest=temp.*pbest+(1-temp).*output.pos;
    pbestval=tmp.*pbestval+(1-tmp).*e;%update the pbest
    [gbestval,tmp]=min(pbestval);
    gbest=pbest(tmp,:);
    gbestrep=repmat(gbest,ps,1);%update the gbest
end
SPV
function [ output ] = SPV( ContiniousX, Job,output)
DiscreteX = zeros(size(ContiniousX));
GrayX = mat2gray(ContiniousX);
for i = 1:size(ContiniousX,1)
    [~,Index] = sort(GrayX(i,:));
    A = Job(Index,:);
    B = A';
    B([2:1:4], :) = [] ;
    DiscreteX(i,:) = B;
end
output.X = DiscreteX;

HGP1
function [ AGVN, CInput ] = AGV( X, problem )
% X = [SEQUENCE]
%[ problem ] = Problem5;
%
ieval = 1;

```



```

output.X(ieval,:) = X;
Discrete_X = output.X(ieval,:);
TravTime = problem.TravTime;
Job = problem.Job;
Gamma = problem.Gamma;
Alpha = problem.Alpha;
ChargingTime = problem.ChargingTime;
Dimension = problem.Dimension ;
InitialAGV = problem.InitialAGV;
ChargeValue = problem.ChargeValue;

CInput.JN = Discrete_X; % job number ;
CInput.ON = Discrete_X; % Operation Number;
CInput.StartPoint = Discrete_X; % Machine Number ;
CInput.MN = Discrete_X; % Machine Number ;
CInput.AGVN = zeros(size(CInput.JN)); % AGV Number which assigned to
load.
CInput.OT = Discrete_X; % Operation Time;
CInput.ExeTime = zeros(size(CInput.JN)); % The time when the operation
started,
AGV start %this time is the time that
from previous position %to move to pick the part

CInput.UnLAGVTime = zeros(size(CInput.JN)); % this the time that take
to AGV go to % the pick up position of
the part % from current position to
CInput.StartPoint

CInput.LAGVTime = zeros(size(CInput.JN)); % This is the time take to
AGV reach % the position of the
operation and % from
CInput.StartPoint(i) to CInput.MN(i)

CInput.SrAGVIdleTime = zeros(size(CInput.JN));% this is the time that
AGV is idle % to unload the part from
previous machine % and move to the new
machine % for new operation.

CInput.StAGVIdleTime = zeros(size(CInput.JN));% this is the time that
AGV is idle % to load the part into
the machine % if the machine is stil
% running for the previous
% operation.

CInput.TotalOpTime = zeros(size(CInput.JN));% This is the total time
required % to finish the operation.

%Calculating the operation number
for i=1:size(CInput.ON,2)
    index = 0;

```

```

        for j=1:i
            if CInput.JN(j) == CInput.JN(i)
                index = index +1;
            end
        end
        CInput.ON(i) = index;
    end

% Importing machine number and operation time for each operation.
for i=1:size(CInput.JN,2)
    for j=1:size(CInput.JN,2)
        if Job(j,1)== CInput.JN(i) && Job(j,2)== CInput.ON(i)
            CInput.MN(i) = Job(j,3);
            CInput.OT(i) = Job(j,4);
        end
    end
end

% producing the starting point of each operation
CInput.StartPoint(1)= 0;
for i=2:size(CInput.JN,2)
    if CInput.ON(i) == 1
        CInput.StartPoint(i) = 0;
    else
        for j=i-1:-1:1
            if CInput.JN(j)== CInput.JN(i)
                CInput.StartPoint(i) = CInput.MN(j);
                break
            end
        end
    end
end

%producing Load AGV Time of travelling AGV
for i=1:size(CInput.JN,2)
    CInput.LAGVTime(i) =
    TravTime(CInput.StartPoint(i)+1,CInput.MN(i)+1);
end

if InitialAGV == 1
    initAGV = ceil (size(CInput.JN,2)/10);
    AGVData.Charge = ones(1, initAGV) .* ChargeValue; %Chrg is the
current AGV's battery level
    AGVData.Position = zeros(1,size(AGVData.Charge,2));% is the
current AGV position
    AGVData.State = ones(1,size(AGVData.Charge,2)); % is the state of
AGV where is it on operation (1) or in charging state (0)
    AGVData.TimeOfCharging = zeros(1,size(AGVData.Charge,2)); % if the
AGVData.State = 0 this shows the time that it has been connected to
charger.
    AGVData.ReadyTime = zeros(1,size(AGVData.Charge,2)); % when thee
agv was ready to perform new task
    AGVData.TimesCharged = zeros(1,size(AGVData.Charge,2)); % how many
times the AGV was charged
elseif InitialAGV == 0
    AGVData.Charge = [ChargeValue]; %Chrg is the current AGV's
battery level
    AGVData.Position = zeros(size(AGVData.Charge,2));% is the current
AGV position
    AGVData.State = ones(size(AGVData.Charge,2)); % is the state of
AGV where is it on operation (1) or in charging state (0)

```

```

AGVData.TimeOfCharging = zeros(size(AGVData.Charge,2)); % if the
AGVData.State = 0 this shows the time that it has been connected to
charger.
AGVData.ReadyTime = zeros(size(AGVData.Charge,2)); % when thee agv
was ready to perform new task
AGVData.TimesCharged = zeros(size(AGVData.Charge,2)); % how many
times the AGV was charged
elseif InitialAGV == 2
    initAGV = max(CInput.JN);
    AGVData.Charge = ones(1, initAGV) .* ChargeValue; %Chрге is the
current AGV's battery level
    AGVData.Position = zeros(1,size(AGVData.Charge,2));% is the
current AGV position
    AGVData.State = ones(1,size(AGVData.Charge,2)); % is the state of
AGV where is it on operation (1) or in charging state (0)
    AGVData.TimeOfCharging = zeros(1,size(AGVData.Charge,2)); % if the
AGVData.State = 0 this shows the time that it has been connected to
charger.
    AGVData.ReadyTime = zeros(1,size(AGVData.Charge,2)); % when thee
agv was ready to perform new task
    AGVData.TimesCharged = zeros(1,size(AGVData.Charge,2)); % how many
times the AGV was charged
end

AGVData.Charge(1) = AGVData.Charge(1) - Gamma *
TravTime(1,CInput.MN(1)+1);
CInput.AGVN(1) = 1;
CInput.ExeTime(1) = 0;
CInput.UnLAGVTime(1) = TravTime(AGVData.Position
(1)+1,CInput.StartPoint(1)+1);
CInput.SrAGVIdleTime(1) = 0;
CInput.StAGVIdleTime(1) = 0;
CInput.TotalOpTime(1) = CInput.OT(1) + CInput.ExeTime(1) +
CInput.UnLAGVTime(1) + ...
    CInput.LAGVTime(1) + CInput.SrAGVIdleTime(1) +
CInput.StAGVIdleTime(1);
CInput.ExeTime(2) = CInput.ExeTime(1) + CInput.UnLAGVTime(1) + ...
    CInput.LAGVTime(1) + CInput.SrAGVIdleTime(1) +
CInput.StAGVIdleTime(1);
AGVData.Position(1) = CInput.MN(1);%AGV position, 0 means at home

for i=2:size(CInput.JN,2)
    % avval tashkhis bede bebin az koja mikhai beri koja
CInput.StartPoint(i) va CInput.MN(i) ro adad bede
    % bebin un AGV agar bekhad bere load kone va unload kone cheghadr
zaman
    % migire, hamin karo vase tamame AGV haye mojud dar system anjam
bede
    % zemne in ke bebin sharjeshun kafi hast ya na

    % agar zamane ye AGV jadid az nesfe AGV haye mojud kamtar bud AGV
% jadid biar to madar
    % ChA = [ChA , ChargeValue];

    StPToHomeCharge = TravTime(CInput.MN(i)+1,1);
    if CInput.ON(i) == 1
        POp = 0;% previous operation related to this job
    else
        for j=i-1:-1:1
            if CInput.JN(j)== CInput.JN(i)
                POp = j; % previous operation related to this job
                break
            end
        end
    end
end

```

```

        end
    end

    for j=1:size(AGVData.Charge,2)
        if AGVData.Charge(j) == ChargeValue && AGVData.Position(j) ==
0 && AGVData.State(j) == 2
            addnew = 0;
            break
        else
            addnew = 1 ;
        end
    end
end

if InitialAGV == 2
    addnew = 0;
end

if addnew
%selecting proper AGV
    AGVCalc.Charge = [AGVData.Charge,ChargeValue];
    AGVCalc.Position = [AGVData.Position , 0];
    AGVCalc.Time = zeros(size(AGVCalc.Charge));
    AGVCalc.select = zeros(size(AGVCalc.Charge));
    selectedAGV = 1 ;
else
    AGVCalc.Charge = [AGVData.Charge];
    AGVCalc.Position = [AGVData.Position];
    AGVCalc.Time = zeros(size(AGVCalc.Charge));
    AGVCalc.select = zeros(size(AGVCalc.Charge));
    selectedAGV = 1 ;
end

%
% %selecting proper AGV
%     AGVCalc.Charge = [AGVData.Charge,ChargeValue];
%     AGVCalc.Position = [AGVData.Position , 0];
%     AGVCalc.Time = zeros(size(AGVCalc.Charge));
%     AGVCalc.select = zeros(size(AGVCalc.Charge));
%     selectedAGV = 1 ;

% Checking the charge of each AGV and charging state of them
    for j=1:size(AGVData.Charge,2)
        if AGVData.State(j) == 0 && AGVData.ReadyTime(j) <=
(CInput.ExeTime (i))
            AGVData.Charge(j) = ChargeValue;
            AGVData.Position(j) = 0;
            AGVData.State(j) = 1;
            AGVData.TimeOfCharging(j) = 0;
            AGVData.TimesCharged (j) = AGVData.TimesCharged (j) + 1;
        end
    end
    for j=1:size(AGVData.Charge,2)
        if AGVData.Charge(j)< ...
(TravTime(AGVData.Position(j)+1,CInput.StartPoint(i)+1) + ...
            TravTime(CInput.StartPoint(i)+1,CInput.MN(i)+1))+ ...
            TravTime(CInput.MN(i)+1,1) * ...
            Gamma ...
            && AGVData.State(j) ~= 0
            AGVData.Charge(j) = AGVData.Charge(j) -
TravTime(AGVData.Position(j)+1,1)*Gamma;
            if AGVData.Charge(j) < 0
                if exist('Error', 'var')
                    Error.AGVData = [Error.AGVData; AGVData];

```

```

Error.AGVSelection = [Error.AGVSelection;
AGVCalc];
Error.CInput = [Error.CInput; CInput];
else
Error.AGVData = AGVData;
Error.AGVSelection = AGVCalc;
Error.CInput = CInput;
end
end
AGVCalc.select(j) = 2;
AGVData.State(j) = 0;
AGVData.TimeOfCharging(j) = CInput.ExeTime(i) +
TravTime(AGVData.Position(j)+1,1);
AGVData.ReadyTime(j) = AGVData.TimeOfCharging(j) +
ChargingTime;
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j=1:size(AGVCalc.Charge,2)
%testing the remainder of charge after operation
AGVCalc.Time(j) =
TravTime(AGVCalc.Position(j)+1,CInput.StartPoint(i)+1) + ...
TravTime(CInput.StartPoint(i)+1,CInput.MN(i)+1) * Gamma;

AGVCalc.Charge(j) = AGVCalc.Charge(j) - StPToHomeCharge -
...
AGVCalc.Time(j) * Gamma ;

if j <= size(AGVData.State,2)
if AGVData.State(j) == 0
AGVCalc.select(j) = 2;
end
end
%Selecting the AGV
if (j > 1 && j < size(AGVCalc.Charge,2) && AGVCalc.select(j) ~=
2) && InitialAGV ~= 2
if AGVCalc.Time(j) < AGVCalc.Time(selectedAGV) ||
AGVCalc.select(selectedAGV) == 2
selectedAGV = j;
end
end
if j == size(AGVCalc.Charge,2) && InitialAGV ~= 2
if Alpha * AGVCalc.Time(j) < AGVCalc.Time(selectedAGV) ||
AGVCalc.select(selectedAGV) == 2
selectedAGV = j;
AGVData.Charge = [AGVData.Charge, ChargeValue];
AGVData.Position = [AGVData.Position, 0];
AGVData.State = [AGVData.State, 1];
AGVData.TimeOfCharging = [AGVData.TimeOfCharging, 0];
AGVData.ReadyTime = [AGVData.ReadyTime, 0];
AGVData.TimesCharged = [AGVData.TimesCharged, 0];
end
end

if (j > 1 && AGVCalc.select(j) ~= 2) && InitialAGV == 2
if AGVCalc.Time(j) < AGVCalc.Time(selectedAGV) ||
AGVCalc.select(selectedAGV) == 2
selectedAGV = j;
end
end
end
if min(AGVCalc.select) == 2
[A, Index] = min(AGVData.ReadyTime);

```

```

        if CInput.ExeTime(i) < A
            CInput.ExeTime(i) = A;
            selectedAGV = Index;
            AGVData.Charge(selectedAGV) = ChargeValue;
            AGVData.Position(selectedAGV) = 0;
            AGVData.State(selectedAGV) = 1;
            AGVData.TimeOfCharging(selectedAGV) = 0;
            AGVData.TimesCharged (selectedAGV) = AGVData.TimesCharged
(selectedAGV) + 1;
        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MachineDelayToUnload = 0;
for j=i-1:-1:1
    if CInput.MN(i) == CInput.MN(j)
        MachineDelayToUnload = CInput.TotalOpTime(j); %Previous
operation of the machine
        break;
    end
end
end

CInput.UnLAGVTime(i) = TravTime(AGVData.Position
(selectedAGV)+1,CInput.StartPoint(i)+1);

if CInput.ON(i) == 1
    CInput.SrAGVIdleTime(i) = 0;
    CInput.StAGVIdleTime(i) = MachineDelayToUnload -
(CInput.ExeTime(i) + ...
    CInput.SrAGVIdleTime(i) + CInput.UnLAGVTime(i) +
CInput.LAGVTime(i));

    if CInput.StAGVIdleTime(i)<0
        CInput.StAGVIdleTime(i) = 0;
    end
else

    % calculate the starting Idle time
    CInput.SrAGVIdleTime(i) = CInput.TotalOpTime(POp) -
(CInput.ExeTime(i) + CInput.UnLAGVTime(i));
    if CInput.SrAGVIdleTime(i)<0
        CInput.SrAGVIdleTime(i) = 0;
    end
    % calculate the Stopp ing Idle time
    CInput.StAGVIdleTime(i) = MachineDelayToUnload -
(CInput.ExeTime(i) + ...
    CInput.SrAGVIdleTime(i) + CInput.UnLAGVTime(i) +
CInput.LAGVTime(i));
    if CInput.StAGVIdleTime(i)<0
        CInput.StAGVIdleTime(i) = 0;
    end
end
end

CInput.TotalOpTime(i) = CInput.OT(i) + CInput.ExeTime(i) +
CInput.UnLAGVTime(i) + ...
    CInput.LAGVTime(i) + CInput.SrAGVIdleTime(i) +
CInput.StAGVIdleTime(i);

AGVData.Charge(selectedAGV) = AGVData.Charge(selectedAGV) - ...
    Gamma * (CInput.UnLAGVTime(i) + CInput.LAGVTime(i));
AGVData.Position(selectedAGV) = CInput.MN(i);
CInput.AGVN(i) = selectedAGV;
if i<size(CInput.JN,2)

```

```

        CInput.ExeTime(i+1) = CInput.ExeTime(i) + CInput.UnLAGVTime(i)
+ ...
        CInput.LAGVTime(i) + CInput.SrAGVIdleTime(i) +
CInput.StAGVIdleTime(i);
    end

    clear AGVCalc;
    DeadEnd = 0;

end
output.Y.Makespan(ieval,1) = max(CInput.TotalOpTime);
output.Y.NAGV(ieval,1) = size(AGVData.Charge,2);
AGVN = CInput.AGVN;
End

```

## HGP 2

```

clc
clear variables

[ problem ] = Problem4;
for w = 1:problem.maxRun
    tic;
    problem.Weighted sum = '+'; % '+' --> makespan + AGV, '*'-->
makespan * AGV,
                                % 'makespan' --> makespan, 'AGV' --> AGV,

    %% Parameter tune
    HGAPSO.Parameter.MaxGen = 200; %Generation,
programmer default=500
    HGAPSO.Parameter.UB = 10;
    HGAPSO.Parameter.LB = 0;
    HGAPSO.Parameter.D = problem.Dimension;
    HGAPSO.Parameter.N = 200; % population,
programmer default=20
    HGAPSO.Parameter.C = [0.01, 0.9]; % [C1 , C2]
    HGAPSO.Parameter.InertiaWeight = [0.01, 0.5]; % [Minimum
inertia weight, Maximum inertia weight,]
    HGAPSO.Parameter.alfa = 0; % Parameter for
linear crossover, 0 by default
    HGAPSO.Parameter.Pc = 0.9; % Crossover
probability, 0.9 by default
    HGAPSO.Parameter.Pm = 0.08; % Mutation
probability, 0.1 by default
    HGAPSO.Parameter.CrP = 0.2; % percentage
iterations with crossover
    HGAPSO.Parameter.Elitism = [1, 0.0, 1.0]; % Elitism for GA
starting and ending percent from maximum iteration
% FOR

Hybrid_GA_PSO
    HGAPSO.Parameter.parallel = 1; % 1 = PSO output
to GA, 0 = GA from old results
    HGAPSO.Parameter.Reconstructor = 0.2; % the percentage
which worst results should be regenerated
    %% RUN
    [HGAPSO.output] = Hybrid_GA_PSO_1(HGAPSO.Parameter, problem);
    HGAPSO.time = toc;
%% -----Report Hybrid GA - PSO -----
    HGAPSO.output.History.Y.Makespan(1,:) = [];
    HGAPSO.output.History.Y.Eval(1,:) = [];
    HGAPSO.output.History.Y.NAGV(1,:) = [];

    [HGAPSO.output.Best.Y.Eval, I] =
min(HGAPSO.output.History.Y.Eval);

```

```

HGAPSO.output.Best.Y.Makespan =
HGAPSO.output.History.Y.Makespan(I);
HGAPSO.output.Best.Y.NAGV = HGAPSO.output.History.Y.NAGV(I);
HGAPSO.output.Best.X = HGAPSO.output.History.X(I,:);
fprintf('The Hybrid GA-PSO results with "%6.0f " function
evaluation      \n', HGAPSO.Parameter.MaxGen * HGAPSO.Parameter.N );
fprintf('Makespan = %9.0f \nNumber of AGV = %3.0f \n',
HGAPSO.output.Best.Y.Makespan, HGAPSO.output.Best.Y.NAGV);
fprintf('time = %4.4f Sec\n', HGAPSO.time);
fprintf('\n-----
\n\n');
HGAPSO.output.Best.Y.Eval = min(HGAPSO.output.History.Y.Eval);
HGAPSO.output.Best.Y.Makespan =
min(HGAPSO.output.History.Y.Makespan);
HGAPSO.output.Best.Y.NAGV = min(HGAPSO.output.History.Y.NAGV);

HGAPSO.output.Worst.Y.Eval = max(HGAPSO.output.History.Y.Eval);
HGAPSO.output.Worst.Y.Makespan =
max(HGAPSO.output.History.Y.Makespan);
HGAPSO.output.Worst.Y.NAGV = max(HGAPSO.output.History.Y.NAGV);

HGAPSO.output.Mean.Y.Eval = mean(HGAPSO.output.History.Y.Eval);
HGAPSO.output.Mean.Y.Makespan =
mean(HGAPSO.output.History.Y.Makespan);
HGAPSO.output.Mean.Y.NAGV = mean(HGAPSO.output.History.Y.NAGV);

HGAPSO.output.ST.Y.Eval = std(HGAPSO.output.History.Y.Eval);
HGAPSO.output.ST.Y.Makespan =
std(HGAPSO.output.History.Y.Makespan);
HGAPSO.output.ST.Y.NAGV = std(HGAPSO.output.History.Y.NAGV);

finalReport.HGAPSO.Eval(w,:) = [HGAPSO.output.Best.Y.Eval,
HGAPSO.output.Worst.Y.Eval, HGAPSO.output.Mean.Y.Eval,
HGAPSO.output.ST.Y.Eval];
finalReport.HGAPSO.Makespan(w,:) = [HGAPSO.output.Best.Y.Makespan,
HGAPSO.output.Worst.Y.Makespan, HGAPSO.output.Mean.Y.Makespan,
HGAPSO.output.ST.Y.Makespan];
finalReport.HGAPSO.NAGV(w,:) = [HGAPSO.output.Best.Y.NAGV,
HGAPSO.output.Worst.Y.NAGV, HGAPSO.output.Mean.Y.NAGV,
HGAPSO.output.ST.Y.NAGV];
%% ----- Save Iterations -----

Record(w).HGAPSO = HGAPSO;
clear HGAPSO
end
%% ----- Summary Hybrid GA PSO -----

[finalReport.excel(7,1),I] = min(finalReport.HGAPSO.Eval(:,1));
finalReport.excel(7,2) = max(finalReport.HGAPSO.Eval(:,2));
finalReport.excel(7,3) = min(finalReport.HGAPSO.Eval(:,3));
finalReport.excel(7,4) = min(finalReport.HGAPSO.Eval(:,4));

finalReport.excel(8,1) = min(finalReport.HGAPSO.Makespan(:,1));
finalReport.excel(8,2) = max(finalReport.HGAPSO.Makespan(:,2));
finalReport.excel(8,3) = min(finalReport.HGAPSO.Makespan(:,3));
finalReport.excel(8,4) = min(finalReport.HGAPSO.Makespan(:,4));

finalReport.excel(9,1) = min(finalReport.HGAPSO.NAGV(:,1));
finalReport.excel(9,2) = max(finalReport.HGAPSO.NAGV(:,2));
finalReport.excel(9,3) = min(finalReport.HGAPSO.NAGV(:,3));
finalReport.excel(9,4) = min(finalReport.HGAPSO.NAGV(:,4));

```



```

HGAPSO = Record(I).HGAPSO;

%% ----- Draw Graph -----
CountHGAPSO = [1:HGAPSO.Parameter.MaxGen];

MAXIMUM = max([max(HGAPSO.output.History.Y.Makespan)]);
DIGIT = ceil(log10(abs(MAXIMUM))/2);
Max_Axe = ceil(MAXIMUM/(10^DIGIT))*10^DIGIT;
MINIMUM = min([min(HGAPSO.output.History.Y.Makespan)]);
DIGIT = ceil(log10(abs(MINIMUM))/2);
Min_Axe = floor(MINIMUM/(10^DIGIT))*10^DIGIT;

%% ----- Plot Hybrid GA-PSO -----
HGAPSO_PLOT = zeros(HGAPSO.Parameter.MaxGen,3);
for i = 1 : HGAPSO.Parameter.MaxGen-1
    HGAPSO_PLOT(i,1) = min(
HGAPSO.output.History.Y.Makespan(((i-1)*HGAPSO.Parameter.N+1) :
(i*HGAPSO.Parameter.N)));
    HGAPSO_PLOT(i,2) = max(
HGAPSO.output.History.Y.Makespan(((i-1)*HGAPSO.Parameter.N+1) :
(i*HGAPSO.Parameter.N)));
    HGAPSO_PLOT(i,3) = sum(
HGAPSO.output.History.Y.Makespan(((i-1)*HGAPSO.Parameter.N+1) :
(i*HGAPSO.Parameter.N))/HGAPSO.Parameter.N;
    end

    HGAPSO_PLOT(HGAPSO.Parameter.MaxGen,1) = min(
HGAPSO.output.History.Y.Makespan(((HGAPSO.Parameter.MaxGen-
1)*HGAPSO.Parameter.N+1) :
(HGAPSO.Parameter.MaxGen*HGAPSO.Parameter.N)));
    HGAPSO_PLOT(HGAPSO.Parameter.MaxGen,2) = max(
HGAPSO.output.History.Y.Makespan(((HGAPSO.Parameter.MaxGen-
1)*HGAPSO.Parameter.N+1) :
(HGAPSO.Parameter.MaxGen*HGAPSO.Parameter.N)));
    HGAPSO_PLOT(HGAPSO.Parameter.MaxGen,3) = sum(
HGAPSO.output.History.Y.Makespan(((HGAPSO.Parameter.MaxGen-
1)*HGAPSO.Parameter.N+1) :
(HGAPSO.Parameter.MaxGen*HGAPSO.Parameter.N))/HGAPSO.Parameter.N;

    figure('Name','Hybrid GA-PSO Minimum','NumberTitle','off')
    plot ( CountHGAPSO, HGAPSO_PLOT(:,1))
    axis([0,HGAPSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])
    figure('Name','Hybrid GA-PSO Maximum','NumberTitle','off')
    plot ( CountHGAPSO, HGAPSO_PLOT(:,2))
    axis([0,HGAPSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])
    figure('Name','Hybrid GA-PSO Mean','NumberTitle','off')
    plot ( CountHGAPSO, HGAPSO_PLOT(:,3))
    axis([0,HGAPSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])

    filename = strcat('Record_HGAPSO_', num2str(sum(clock)*1000),
'.mat');
    save(filename,'Record');

```

### MAIN (including four algorithms together)

```

clc
% clear variables
[ problem ] = Problem4;
for run = 1:problem.maxRun

    fprintf('----- Run #%2.0f ----- \n',
run );

```

```

        problem.Weighted sum = '+maxmakespan';    % '+' --> makespan + AGV,
        '*'--> makespan * AGV,
                                % 'makespan' --> makespan, 'AGV' --> AGV,
%% -----PSO-----
    tic;
    PSO.Parameter.MaxGen = 100;                %Generation,
programmer default=500
    PSO.Parameter.UB = 10;                    %upper boundary
    PSO.Parameter.LB = 0;                    %Lower boundary
    PSO.Parameter.N = 100;                    %population,
programmer default=20
    PSO.Parameter.D = problem.Dimension;
    PSO.Parameter.C = [2, 2];                %[C1, C2,], programmer
default=[2, 2]
    PSO.Parameter.InertiaWeight = [0.2, 0.6]; % [Minimum inertia
weight, Maximum inertia weight,], default[0.2, 0.6]

    [PSO.output] = PSO_func(PSO.Parameter, problem);
    PSO.time = toc;
%% -----GA-----
    tic;
    GA.Parameter.MaxGen = 100;                %Generation, programmer
default=100
    GA.Parameter.UB = 10;                    %upper boundary
    GA.Parameter.LB = 0;                    %Lower boundary
    GA.Parameter.D = problem.Dimension;
    GA.Parameter.N = 100;                    %population, programmer
default=100
    GA.Parameter.alfa = 0;                  % Parameter for linear
crossover, 0 by default
    GA.Parameter.Pc = 0.2;                  %Crossover probability,
0.9 by default
    GA.Parameter.Pm = 0.03;                 % Mutation probability,
0.1 by default

    [GA.output] = GA_func(GA.Parameter, problem);
    GA.time = toc;
%% ----- HGP1 -----
    tic;
    %% Parameter tune
    HGAPSO.Parameter.MaxGen = 100;          %Generation,
programmer default=500
    HGAPSO.Parameter.UB = 10;
    HGAPSO.Parameter.LB = 0;
    HGAPSO.Parameter.D = problem.Dimension;
    HGAPSO.Parameter.N = 100;              % population,
programmer default=20
    HGAPSO.Parameter.C = [0.01, 0.9];      % [C1 , C2]
    HGAPSO.Parameter.InertiaWeight = [0.01, 0.5]; % [Minimum
inertia weight, Maximum inertia weight,]
    HGAPSO.Parameter.alfa = 0;             % Parameter for
linear crossover, 0 by default
    HGAPSO.Parameter.Pc = 0.9;             % Crossover
probability, 0.9 by default
    HGAPSO.Parameter.Pm = 0.08;           % Mutation
probability, 0.1 by default
    HGAPSO.Parameter.CrP = 0.2;           % percentage
iterations with crossover
    HGAPSO.Parameter.Elitism = [1, 0.0, 1.0]; % Elitism for GA
starting and ending percent from maximum iteration
    % FOR
Hybrid_GA_PSO

```

```

HGAPSO.Parameter.parallel = 1; % 1 = PSO output
to GA, 0 = GA from old results
HGAPSO.Parameter.Reconstructor = 0.2; % the percentage
which worst results should be regenerated

[HGAPSO.output] = Hybrid_GA_PSO_1(HGAPSO.Parameter, problem);
HGAPSO.time = toc;
%% ----- HGP2-----
tic;
%% Parameter tune
HGAPSO.Parameter.MaxGen = 100; %Generation,
programmer default=500
HGAPSO.Parameter.UB = 10;
HGAPSO.Parameter.LB = 0;
HGAPSO.Parameter.D = problem.Dimension;
HGAPSO.Parameter.N = 100; % population,
programmer default=20
HGAPSO.Parameter.C = [0.01, 0.9]; % [C1 , C2]
HGAPSO.Parameter.InertiaWeight = [0.01, 0.5]; % [Minimum
inertia weight, Maximum inertia weight,]
HGAPSO.Parameter.alfa = 0; % Parameter for
linear crossover, 0 by default
HGAPSO.Parameter.Pc = 0.9; % Crossover
probability, 0.9 by default
HGAPSO.Parameter.Pm = 0.08; % Mutation
probability, 0.1 by default
HGAPSO.Parameter.CrP = 0.2; % percentage
iterations with crossover
HGAPSO.Parameter.Elitism = [1, 0.0, 1.0]; % Elitism for GA
starting and ending percent from maximum iteration % FOR
Hybrid_GA_PSO

HGAPSO.Parameter.parallel = 1; % 1 = PSO output
to GA, 0 = GA from old results
HGAPSO.Parameter.Reconstructor = 0.2; % the percentage
which worst results should be regenerated

[HGAPSO.output] = Hybrid_GA_PSO_1(HGAPSO.Parameter, problem);
HGAPSO.time = toc;
%% -----Report PSO-----
PSO.output.History.Y.Makespan(1,:) = [];
PSO.output.History.Y.Eval(1,:) = [];
PSO.output.History.Y.NAGV(1,:) = [];
% PSO.output.History.X(1,:) = [];

%% -----Report GA-----
GA.output.History.Y.Makespan(1,:) = [];
GA.output.History.Y.Eval(1,:) = [];
GA.output.History.Y.NAGV(1,:) = [];
% GA.output.History.X(1,:) = [];

%% -----Report Hybrid GA - PSO -----
HGAPSO.output.History.Y.Makespan(1,:) = [];
HGAPSO.output.History.Y.Eval(1,:) = [];
HGAPSO.output.History.Y.NAGV(1,:) = [];
% HGAPSO.output.History.X(1,:) = [];

%% ----- Save Iterations -----
Record(run).GA = GA;
Record(run).PSO = PSO;
Record(run).HGAPSO = HGAPSO;
Record(run).time = GA.time + PSO.time + HGAPSO.time;

```

```

%% ----- Clear Variables -----
    if problem.maxRun > 1
        clear GA PSO HGAPSO
    end
    fprintf('Total Runing time for this Run: %4.4f
Sec\n\n',Record(run).time);
    fprintf('GA Runing time for this Run: %4.4f
Sec\n\n',Record(run).GA.time);
    fprintf('PSO Runing time for this Run: %4.4f
Sec\n\n',Record(run).PSO.time );
    fprintf('HGAPSO Runing time for this Run: %4.4f
Sec\n\n',Record(run).HGAPSO.time);

end
%% ----- Save Data -----
%
% filename = strcat('Record_', num2str(sum(clock)*1000), '.mat');
% save(filename, 'Record');

%% ----- Reporting -----
Record = ReportRecord( Record );
% DrawGraph( Record, finalReport, maxRun);
% DrawGraohMakespan( Record, finalReport, problem);
% [Best.GA.Y,I] = min(Record.GA.output.History.Y.Eval);
% Best.GA.X = Record.GA.output.History.X(I,:);
% Best.GA.HistoryIndex = I;
%
% [Best.PSO.Y,I] = min(Record.PSO.output.History.Y.Eval);
% Best.PSO.X = Record.PSO.output.History.X(I,:);
% Best.PSO.HistoryIndex = I;
%
% [Best.HGAPSO.Y,I] = min(Record.GA.output.History.Y.Eval);
% Best.HGAPSO.X = Record.HGAPSO.output.History.X(I,:);
% Best.HGAPSO.HistoryIndex = I;

PROBLEM
function [ problem ] = Problem4( )

% input = [4 1 5 3 6 2 4 3 1 3 1 6 4 3 2 3 1 6 2 4 1 3 5 4 3 2 1 2 1 4
5 3 2 6 1 6];
% input = [3 4 1 1 2 2 1 2 1 3 3 3];

% Table 1
% Min   L/U  M1   M2   M3   M4   M5   M6
% L/U   0    6    8    10   12   17   19
% M1    15   0    2    5    7    12   14
% M2    13   18   0    3    5    10   12
% M3    10   15   17   0    2    7    9
% M4    8    13   15   18   0    5    7
% M5    3    8    10   13   15   0    2
% M6    1    6    8    11   13   18   0
%
%
%
% Table 2
% Code  Operation   Machine Operation-time
% 1      11           2           30
% 1      12           1           21
% 1      13           5           24
% 1      14           6           27
%

```

```

% 2      21      1      15
% 2      22      4      24
% 2      23      6      13
%
% 3      31      1      16
% 3      32      2      21
% 3      33      3      3
% 3      34      4      14
% 3      35      6      25

```

```

% TravTime = [ 0   6   8  10  12  17  19
%              15   0   2   5   7  12  14
%              13  18   0   3   5  10  12
%              10  15  17   0   2   7   9
%              8   13  15  18   0   5   7
%              3   8  10  13  15   0   2
%              1   6   8  11  13  18   0  ];

```

```

problem.TravTime = [ 0  6 18 28 42 36 38 17 50 63 37 24 10
%                   34 0 12 22 36 50 52 31 64 77 71 58 44
%                   22 28 0 10 24 38 40 19 52 65 59 46 32
%                   34 40 52 0 14 28 48 31 42 55 71 58 44
%                   34 40 52 42 0 14 34 31 28 41 71 58 44
%                   58 64 76 66 80 0 20 41 14 27 61 48 68
%                   38 46 58 46 60 54 0 21 12 25 41 28 48
%                   17 23 35 25 39 33 21 0 33 46 40 27 27
%                   64 70 82 72 86 80 44 47 0 13 67 54 74
%                   51 57 69 59 73 67 31 34 43 0 54 41 61
%                   41 47 59 49 63 57 21 24 33 46 0 31 51
%                   54 60 72 62 76 70 34 37 46 59 13 0 64
%                   44 50 62 52 66 60 28 27 40 53 27 14 0  ];

```

```

% Job =      [ 1      1      2      30
%              1      2      1      21
%              1      3      5      24
%              1      4      6      27
%              2      1      1      15
%              2      2      4      24
%              2      3      6      13
%              3      1      1      16
%              3      2      2      21
%              3      3      3      3
%              3      4      4      14
%              3      5      6      25  ];

```

```

problem.Job = [ 1      1      2      37
%              1      2      6      33
%              1      3      5      34
%              1      4      8      35
%              1      5      1      23
%              1      6      12     34
%              1      7      7      37
%              1      8      5      26
%              2      1      3      23
%              2      2      4      26
%              2      3      6      27
%              2      4      11     25
%              2      5      10     34
%              2      6      9      23
%              3      1      1      26
%              3      2      2      25

```

3	3	10	31
3	4	4	24
3	5	6	25
3	6	7	13
3	7	8	14
3	8	11	23
4	1	1	16
4	2	7	11
4	3	5	23
4	4	12	34
4	5	6	25
4	6	8	13
5	1	1	16
5	2	7	11
5	3	9	31
6	1	3	26
6	2	2	31
6	3	10	23
6	4	4	24
6	5	11	35];

```
problem.Dimension = size(problem.Job,1);
```

```
problem.Gamma = 1; %the ratio of energy consumption to the time
problem.Alpha = 1.5;
problem.ChargingTime = 40;
problem.ChargeValue = 200;
problem.InitialAGV = 1;% 0 = starrrt with one AGV, 1 = Start with
1/10th of genes, 2 = the number of AGVs is equal to Number of Jobs
problem.maxRun = 30;
```

### **FITNESS FUNCTION**

```
function [ output ] = Fitness(ieval, problem, output)
```

```
Discrete_X = output.X(ieval,:);
TravTime = problem.TravTime;
Job = problem.Job;
Gamma = problem.Gamma;
Alpha = problem.Alpha;
ChargingTime = problem.ChargingTime;
Dimension = problem.Dimension ;
InitialAGV = problem.InitialAGV;
ChargeValue = problem.ChargeValue;
```

```
CInput.JN = Discrete_X; % job number ;
CInput.ON = Discrete_X; % Operation Number;
CInput.StartPoint = Discrete_X; % Machine Number ;
CInput.MN = Discrete_X; % Machine Number ;
CInput.AGVN = zeros(size(CInput.JN)); % AGV Number which assigned to
load.
CInput.OT = Discrete_X; % Operation Time;
CInput.ExeTime = zeros(size(CInput.JN)); % The time when the operation
started,
AGV start %this time is the time that
from previous position %to move to pick the part

CInput.UnLAGVTime = zeros(size(CInput.JN)); % this the time that take
to AGV go to
the part % the pick up position of
```

```

CInput.StartPoint                                     % from current position to

CInput.LAGVTime = zeros(size(CInput.JN));           % This is the time take to
AGV reach                                           % the position of the
operation and                                       % from
CInput.StartPoint(i) to CInput.MN(i)

CInput.SrAGVIdleTime = zeros(size(CInput.JN));% this is the time that
AGV is idle                                       % to unload the part from
previous machine                                  % and move to the new
machine                                           % for new operation.

CInput.StAGVIdleTime = zeros(size(CInput.JN));% this is the time that
AGV is idle                                       % to load the part into
the machine                                       % if the machine is stil
                                                % running for the previous
                                                % operation.

CInput.TotalOpTime = zeros(size(CInput.JN));% This is the total time
required                                           % to finish the operation.

%Calculating the operation number
for i=1:size(CInput.ON,2)
    index = 0;
    for j=1:i
        if CInput.JN(j) == CInput.JN(i)
            index = index +1;
        end
    end
    CInput.ON(i) = index;
end

% Importing machine number and operation time for each operatiion.
for i=1:size(CInput.JN,2)
    for j=1:size(CInput.JN,2)
        if Job(j,1)== CInput.JN(i) && Job(j,2)== CInput.ON(i)
            CInput.MN(i) = Job(j,3);
            CInput.OT(i) = Job(j,4);
        end
    end
end

% producing the starting point of each operation
CInput.StartPoint(1)= 0;
for i=2:size(CInput.JN,2)
    if CInput.ON(i) == 1
        CInput.StartPoint(i) = 0;
    else
        for j=i-1:-1:1
            if CInput.JN(j)== CInput.JN(i)
                CInput.StartPoint(i) = CInput.MN(j);
                break
            end
        end
    end
end

```

```

        end
    end
end

%producing Load AGV Time of travelling AGV
for i=1:size(CInput.JN,2)
    CInput.LAGVTime(i) =
TravTime(CInput.StartPoint(i)+1,CInput.MN(i)+1);
end

if InitialAGV == 1
    initAGV = ceil (size(CInput.JN,2)/10);
    AGVData.Charge = ones(1, initAGV) .* ChargeValue; %Chrg is the
current AGV's battery level
    AGVData.Position = zeros(1,size(AGVData.Charge,2));% is the
current AGV position
    AGVData.State = ones(1,size(AGVData.Charge,2)); % is the state of
AGV where is it on operation (1) or in charging state (0)
    AGVData.TimeOfCharging = zeros(1,size(AGVData.Charge,2)); % if the
AGVData.State = 0 this shows the time that it has been connected to
charger.
    AGVData.ReadyTime = zeros(1,size(AGVData.Charge,2)); % when thee
agv was ready to perform new task
    AGVData.TimesCharged = zeros(1,size(AGVData.Charge,2)); % how many
times the AGV was charged
elseif InitialAGV == 0
    AGVData.Charge = [ChargeValue]; %Chrg is the current AGV's
battery level
    AGVData.Position = zeros(size(AGVData.Charge,2));% is the current
AGV position
    AGVData.State = ones(size(AGVData.Charge,2)); % is the state of
AGV where is it on operation (1) or in charging state (0)
    AGVData.TimeOfCharging = zeros(size(AGVData.Charge,2)); % if the
AGVData.State = 0 this shows the time that it has been connected to
charger.
    AGVData.ReadyTime = zeros(size(AGVData.Charge,2)); % when thee agv
was ready to perform new task
    AGVData.TimesCharged = zeros(size(AGVData.Charge,2)); % how many
times the AGV was charged
elseif InitialAGV == 2
    initAGV = max(CInput.JN);
    AGVData.Charge = ones(1, initAGV) .* ChargeValue; %Chrg is the
current AGV's battery level
    AGVData.Position = zeros(1,size(AGVData.Charge,2));% is the
current AGV position
    AGVData.State = ones(1,size(AGVData.Charge,2)); % is the state of
AGV where is it on operation (1) or in charging state (0)
    AGVData.TimeOfCharging = zeros(1,size(AGVData.Charge,2)); % if the
AGVData.State = 0 this shows the time that it has been connected to
charger.
    AGVData.ReadyTime = zeros(1,size(AGVData.Charge,2)); % when thee
agv was ready to perform new task
    AGVData.TimesCharged = zeros(1,size(AGVData.Charge,2)); % how many
times the AGV was charged
end

AGVData.Charge(1) = AGVData.Charge(1) - Gamma *
TravTime(1,CInput.MN(1)+1);
CInput.AGVN(1) = 1;
CInput.ExeTime(1) = 0;
CInput.UnLAGVTime(1) = TravTime(AGVData.Position
(1)+1,CInput.StartPoint(1)+1);

```



```

CInput.SrAGVIdleTime(1) = 0;
CInput.StAGVIdleTime(1) = 0;
CInput.TotalOpTime(1) = CInput.OT(1) + CInput.ExeTime(1) +
CInput.UnLAGVTime(1) + ...
    CInput.LAGVTime(1) + CInput.SrAGVIdleTime(1) +
CInput.StAGVIdleTime(1);
CInput.ExeTime(2) = CInput.ExeTime(1) + CInput.UnLAGVTime(1) + ...
    CInput.LAGVTime(1) + CInput.SrAGVIdleTime(1) +
CInput.StAGVIdleTime(1);
AGVData.Position(1) = CInput.MN(1);%AGV position, 0 means at home

for i=2:size(CInput.JN,2)
    % avval tashkhis bede bebin az koja mikhai beri koja
CInput.StartPoint(i) va CInput.MN(i) ro adad bede
    % bebin un AGV agar bekhad bere load kone va unload kone cheghadr
zaman
    % migire, hamin karo vase tamame AGV haye mojood dar system anjam
bede
    % zemne in ke bebin sharjeshun kafi hast ya na

    % agar zamane ye AGV jadid az nesfe AGV haye mojood kamtar bud AGV
    % jadid biar to madar
    % ChA = [ChA , ChargeValue];

    StPToHomeCharge = TravTime(CInput.MN(i)+1,1);
    if CInput.ON(i) == 1
        POP = 0;% previous operation related to this job
    else
        for j=i-1:-1:1
            if CInput.JN(j)== CInput.JN(i)
                POP = j; % previous operation related to this job
                break
            end
        end
    end
end

    for j=1:size(AGVData.Charge,2)
        if AGVData.Charge(j) == ChargeValue && AGVData.Position(j) ==
0 && AGVData.State(j) == 2
            addnew = 0;
            break
        else
            addnew = 1 ;
        end
    end
end

    if InitialAGV == 2
        addnew = 0;
    end

    if addnew
        %selecting proper AGV
        AGVCalc.Charge = [AGVData.Charge,ChargeValue];
        AGVCalc.Position = [AGVData.Position , 0];
        AGVCalc.Time = zeros(size(AGVCalc.Charge));
        AGVCalc.select = zeros(size(AGVCalc.Charge));
        selectedAGV = 1 ;
    else
        AGVCalc.Charge = [AGVData.Charge];
        AGVCalc.Position = [AGVData.Position];
        AGVCalc.Time = zeros(size(AGVCalc.Charge));
        AGVCalc.select = zeros(size(AGVCalc.Charge));
        selectedAGV = 1 ;
    end
end

```

```

end

% %selecting proper AGV
%   AGVCalc.Charge = [AGVData.Charge,ChargeValue];
%   AGVCalc.Position = [AGVData.Position , 0];
%   AGVCalc.Time = zeros(size(AGVCalc.Charge));
%   AGVCalc.select = zeros(size(AGVCalc.Charge));
%   selectedAGV = 1 ;

% Checking the charge of each AGV and charging state of them
for j=1:size(AGVData.Charge,2)
    if AGVData.State(j) == 0 && AGVData.ReadyTime(j) <=
(CInput.ExeTime (i))
        AGVData.Charge(j) = ChargeValue;
        AGVData.Position(j) = 0;
        AGVData.State(j) = 1;
        AGVData.TimeOfCharging(j) = 0;
        AGVData.TimesCharged (j) = AGVData.TimesCharged (j) + 1;
    end
end
for j=1:size(AGVData.Charge,2)
    if AGVData.Charge(j)< ...

(TravTime(AGVData.Position(j)+1,CInput.StartPoint(i)+1) + ...
    TravTime(CInput.StartPoint(i)+1,CInput.MN(i)+1))+ ...
    TravTime(CInput.MN(i)+1,1) * ...
    Gamma ...
    && AGVData.State(j) ~= 0
    AGVData.Charge(j) = AGVData.Charge(j) -
TravTime(AGVData.Position(j)+1,1)*Gamma;
    if AGVData.Charge(j) < 0
        if exist('Error', 'var')
            Error.AGVData = [Error.AGVData; AGVData];
            Error.AGVSelection = [Error.AGVSelection;
AGVCalc];

            Error.CInput = [Error.CInput; CInput];
        else
            Error.AGVData = AGVData;
            Error.AGVSelection = AGVCalc;
            Error.CInput = CInput;
        end
    end
    end
    AGVCalc.select(j) = 2;
    AGVData.State(j) = 0;
    AGVData.TimeOfCharging(j) = CInput.ExeTime(i) +
TravTime(AGVData.Position(j)+1,1);
    AGVData.ReadyTime (j) = AGVData.TimeOfCharging(j) +
ChargingTime;

    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j=1:size(AGVCalc.Charge,2)
    %testing the remainder of charge after operation
    AGVCalc.Time(j) =
TravTime(AGVCalc.Position(j)+1,CInput.StartPoint(i)+1) + ...
    TravTime(CInput.StartPoint(i)+1,CInput.MN(i)+1) * Gamma;

    AGVCalc.Charge (j) = AGVCalc.Charge (j) - StPToHomeCharge -
...
    AGVCalc.Time(j)* Gamma ;

    if j <= size(AGVData.State,2)

```

```

        if AGVData.State(j) == 0
            AGVCalc.select(j) = 2;
        end
    end
    end
    %Selecting the AGV
    if (j > 1 && j < size(AGVCalc.Charge,2) && AGVCalc.select(j) ~=
2 ) && InitialAGV ~= 2
        if AGVCalc.Time(j) < AGVCalc.Time(selectedAGV) ||
AGVCalc.select(selectedAGV) == 2
            selectedAGV = j;
        end
    end
    end
    if j == size(AGVCalc.Charge,2) && InitialAGV ~= 2
        if Alpha * AGVCalc.Time(j) < AGVCalc.Time(selectedAGV) ||
AGVCalc.select(selectedAGV) == 2
            selectedAGV = j;
            AGVData.Charge = [AGVData.Charge, ChargeValue];
            AGVData.Position = [AGVData.Position, 0];
            AGVData.State = [AGVData.State, 1];
            AGVData.TimeOfCharging = [AGVData.TimeOfCharging, 0];
            AGVData.ReadyTime = [AGVData.ReadyTime, 0];
            AGVData.TimesCharged = [AGVData.TimesCharged, 0];
        end
    end
    end

    if (j > 1 && AGVCalc.select(j) ~= 2 ) && InitialAGV == 2
        if AGVCalc.Time(j) < AGVCalc.Time(selectedAGV) ||
AGVCalc.select(selectedAGV) == 2
            selectedAGV = j;
        end
    end
    end
    end
    if min(AGVCalc.select)== 2
        [A, Index] = min(AGVData.ReadyTime);
        if CInput.ExeTime(i) < A
            CInput.ExeTime(i) = A;
            selectedAGV = Index;
            AGVData.Charge(selectedAGV) = ChargeValue;
            AGVData.Position(selectedAGV) = 0;
            AGVData.State(selectedAGV) = 1;
            AGVData.TimeOfCharging(selectedAGV) = 0;
            AGVData.TimesCharged (selectedAGV) = AGVData.TimesCharged
(selectedAGV) + 1;
        end
    end
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    MachineDelayToUnload = 0;
    for j=i-1:-1:1
        if CInput.MN(i) == CInput.MN(j)
            MachineDelayToUnload = CInput.TotalOpTime(j); %Previous
operation of the machine
            break;
        end
    end
    end

    CInput.UnLAGVTime(i) = TravTime(AGVData.Position
(selectedAGV)+1,CInput.StartPoint(i)+1);

    if CInput.ON(i) == 1
        CInput.SrAGVIdleTime(i) = 0;
        CInput.StAGVIdleTime(i) = MachineDelayToUnload -
(CInput.ExeTime(i) + ...

```

```

        CInput.SrAGVIdleTime(i) + CInput.UnLAGVTime(i) +
CInput.LAGVTime(i));

        if CInput.StAGVIdleTime(i)<0
            CInput.StAGVIdleTime(i) = 0;
        end
    else

        % calculate the starting Idle time
        CInput.SrAGVIdleTime(i) = CInput.TotalOpTime(POp) -
(CInput.ExeTime(i) + CInput.UnLAGVTime(i));
        if CInput.SrAGVIdleTime(i)<0
            CInput.SrAGVIdleTime(i) = 0;
        end
        % calculate the Stopp ing Idle time
        CInput.StAGVIdleTime(i) = MachineDelayToUnload -
(CInput.ExeTime(i) + ...
            CInput.SrAGVIdleTime(i) + CInput.UnLAGVTime(i) +
CInput.LAGVTime(i));
        if CInput.StAGVIdleTime(i)<0
            CInput.StAGVIdleTime(i) = 0;
        end
    end

    CInput.TotalOpTime(i) = CInput.OT(i) + CInput.ExeTime(i) +
CInput.UnLAGVTime(i) + ...
        CInput.LAGVTime(i) + CInput.SrAGVIdleTime(i) +
CInput.StAGVIdleTime(i);

    AGVData.Charge(selectedAGV) = AGVData.Charge(selectedAGV) - ...
        Gamma * (CInput.UnLAGVTime(i) + CInput.LAGVTime(i));
    AGVData.Position(selectedAGV) = CInput.MN(i);
    CInput.AGVN(i) = selectedAGV;
    if i<size(CInput.JN,2)
        CInput.ExeTime(i+1) = CInput.ExeTime(i) + CInput.UnLAGVTime(i)
+ ...
            CInput.LAGVTime(i) + CInput.SrAGVIdleTime(i) +
CInput.StAGVIdleTime(i);
    end

    clear AGVCalc;
    DeadEnd = 0;

end
output.Y.Makespan(ieval,1) = max(CInput.TotalOpTime);
output.Y.NAGV(ieval,1) = size(AGVData.Charge,2);
output.AssignedAGV(ieval,:) = (CInput.AGVN);
if strcmp(problem.Weighted sum, '+')
    output.Y.Eval(ieval,1) = (output.Y.Makespan(ieval) +
output.Y.NAGV(ieval))/2;
end
if strcmp(problem.Weighted sum, '*')
    output.Y.Eval(ieval,1) = output.Y.Makespan(ieval) *
output.Y.NAGV(ieval);
end
if strcmp(problem.Weighted sum, 'makespan')
    output.Y.Eval(ieval,1) = output.Y.Makespan(ieval);
end
if strcmp(problem.Weighted sum, '+maxmakespan')
    maxtravel = sum(sum(problem.TravTime));
    maxoperation = sum(problem.Job(:,4));
    maxAGV = size(problem.Job,1);
    ratio =(maxtravel + maxoperation)/maxAGV;%303/maxAGV;%426/maxAGV;

```

```

        output.Y.Eval(ieval,1) = ((2/3).*output.Y.Makespan(ieval) +
(1/3).*output.Y.NAGV(ieval).*ratio );
end

if exist('Error', 'var')
    output.Error = Error;
end

AGV
function [ AGVN, CInput ] = AGV( X, problem )
% X = [SEQUENCE]
%[ problem ] = Problem5;
%
ieval = 1;
output.X(ieval,:) = X;
Discrete_X = output.X(ieval,:);
TravTime = problem.TravTime;
Job = problem.Job;
Gamma = problem.Gamma;
Alpha = problem.Alpha;
ChargingTime = problem.ChargingTime;
Dimension = problem.Dimension ;
InitialAGV = problem.InitialAGV;
ChargeValue = problem.ChargeValue;

CInput.JN = Discrete_X; % job number ;
CInput.ON = Discrete_X; % Operation Number;
CInput.StartPoint = Discrete_X; % Machine Number ;
CInput.MN = Discrete_X; % Machine Number ;
CInput.AGVN = zeros(size(CInput.JN)); % AGV Number which assigned to
load.
CInput.OT = Discrete_X; % Operation Time;
CInput.ExeTime = zeros(size(CInput.JN)); % The time when the operation
started,
%this time is the time that
AGV start
%to move to pick the part
from previous position

CInput.UnLAGVTime = zeros(size(CInput.JN)); % this the time that take
to AGV go to
% the pick up position of
the part
% from current position to
CInput.StartPoint

CInput.LAGVTime = zeros(size(CInput.JN)); % This is the time take to
AGV reach
% the position of the
operation and
% from
CInput.StartPoint(i) to CInput.MN(i)

CInput.SrAGVIdleTime = zeros(size(CInput.JN));% this is the time that
AGV is idle
% to unload the part from
previous machine
% and move to the new
machine
% for new operation.

CInput.StAGVIdleTime = zeros(size(CInput.JN));% this is the time that
AGV is idle

```

```

the machine                                     % to load the part into
                                                % if the machine is stil
                                                % running for the previous
                                                % operation.

CInput.TotalOpTime = zeros(size(CInput.JN));% This is the total time
required
                                                % to finish the operation.

%Calculating the operation number
for i=1:size(CInput.ON,2)
    index = 0;
    for j=1:i
        if CInput.JN(j) == CInput.JN(i)
            index = index +1;
        end
    end
    CInput.ON(i) = index;
end

% Importing machine number and operation time for each operation.
for i=1:size(CInput.JN,2)
    for j=1:size(CInput.JN,2)
        if Job(j,1)== CInput.JN(i) && Job(j,2)== CInput.ON(i)
            CInput.MN(i) = Job(j,3);
            CInput.OT(i) = Job(j,4);
        end
    end
end

% producing the starting point of each operation
CInput.StartPoint(1)= 0;
for i=2:size(CInput.JN,2)
    if CInput.ON(i) == 1
        CInput.StartPoint(i) = 0;
    else
        for j=i-1:-1:1
            if CInput.JN(j)== CInput.JN(i)
                CInput.StartPoint(i) = CInput.MN(j);
                break
            end
        end
    end
end

%producing Load AGV Time of travelling AGV
for i=1:size(CInput.JN,2)
    CInput.LAGVTime(i) =
    TravTime(CInput.StartPoint(i)+1,CInput.MN(i)+1);
end

if InitialAGV == 1
    initAGV = ceil (size(CInput.JN,2)/10);
    AGVData.Charge = ones(1, initAGV) .* ChargeValue; %Chrg is the
current AGV's battery level
    AGVData.Position = zeros(1,size(AGVData.Charge,2));% is the
current AGV position
    AGVData.State = ones(1,size(AGVData.Charge,2)); % is the state of
AGV where is it on operation (1) or in charging state (0)
    AGVData.TimeOfCharging = zeros(1,size(AGVData.Charge,2)); % if the
AGVData.State = 0 this shows the time that it has been connected to
charger.

```

```

    AGVData.ReadyTime = zeros(1,size(AGVData.Charge,2)); % when thee
agv was ready to perform new task
    AGVData.TimesCharged = zeros(1,size(AGVData.Charge,2)); % how many
times the AGV was charged
elseif InitialAGV == 0
    AGVData.Charge = [ChargeValue]; %Chrg is the current AGV's
battery level
    AGVData.Position = zeros(size(AGVData.Charge,2));% is the current
AGV position
    AGVData.State = ones(size(AGVData.Charge,2)); % is the state of
AGV where is it on operation (1) or in charging state (0)
    AGVData.TimeOfCharging = zeros(size(AGVData.Charge,2)); % if the
AGVData.State = 0 this shows the time that it has been connected to
charger.
    AGVData.ReadyTime = zeros(size(AGVData.Charge,2)); % when thee agv
was ready to perform new task
    AGVData.TimesCharged = zeros(size(AGVData.Charge,2)); % how many
times the AGV was charged
elseif InitialAGV == 2
    initAGV = max(CInput.JN);
    AGVData.Charge = ones(1, initAGV) .* ChargeValue; %Chrg is the
current AGV's battery level
    AGVData.Position = zeros(1,size(AGVData.Charge,2));% is the
current AGV position
    AGVData.State = ones(1,size(AGVData.Charge,2)); % is the state of
AGV where is it on operation (1) or in charging state (0)
    AGVData.TimeOfCharging = zeros(1,size(AGVData.Charge,2)); % if the
AGVData.State = 0 this shows the time that it has been connected to
charger.
    AGVData.ReadyTime = zeros(1,size(AGVData.Charge,2)); % when thee
agv was ready to perform new task
    AGVData.TimesCharged = zeros(1,size(AGVData.Charge,2)); % how many
times the AGV was charged
end

AGVData.Charge(1) = AGVData.Charge(1) - Gamma *
TravTime(1,CInput.MN(1)+1);
CInput.AGVN(1) = 1;
CInput.ExeTime(1) = 0;
CInput.UnLAGVTime(1) = TravTime(AGVData.Position
(1)+1,CInput.StartPoint(1)+1);
CInput.SrAGVIdleTime(1) = 0;
CInput.StAGVIdleTime(1) = 0;
CInput.TotalOpTime(1) = CInput.OT(1) + CInput.ExeTime(1) +
CInput.UnLAGVTime(1) + ...
    CInput.LAGVTime(1) + CInput.SrAGVIdleTime(1) +
CInput.StAGVIdleTime(1);
CInput.ExeTime(2) = CInput.ExeTime(1) + CInput.UnLAGVTime(1) + ...
    CInput.LAGVTime(1) + CInput.SrAGVIdleTime(1) +
CInput.StAGVIdleTime(1);
AGVData.Position(1) = CInput.MN(1);%AGV position, 0 means at home

for i=2:size(CInput.JN,2)
    % avval tashkhis bede bebin az koja mikhai beri koja
CInput.StartPoint(i) va CInput.MN(i) ro adad bede
    % bebin un AGV agar bekhad bere load kone va unload kone cheghadr
zaman
    % migire, hamin karo vase tamame AGV haye mojood dar system anjam
bede
    % zemne in ke bebin sharjeshun kafi hast ya na

    % agar zamane ye AGV jadid az nesfe AGV haye mojood kamtar bud AGV
% jadid biar to madar

```

```

% ChA = [ChA , ChargeValue];
%

StPToHomeCharge = TravTime(CInput.MN(i)+1,1);
if CInput.ON(i) == 1
    POP = 0;% previous operation related to this job
else
    for j=i-1:-1:1
        if CInput.JN(j)== CInput.JN(i)
            POP = j; % previous operation related to this job
            break
        end
    end
end

for j=1:size(AGVData.Charge,2)
    if AGVData.Charge(j) == ChargeValue && AGVData.Position(j) ==
0 && AGVData.State(j) == 2
        addnew = 0;
        break
    else
        addnew = 1 ;
    end
end

if InitialAGV == 2
    addnew = 0;
end

if addnew
%selecting proper AGV
    AGVCalc.Charge = [AGVData.Charge,ChargeValue];
    AGVCalc.Position = [AGVData.Position , 0];
    AGVCalc.Time = zeros(size(AGVCalc.Charge));
    AGVCalc.select = zeros(size(AGVCalc.Charge));
    selectedAGV = 1 ;
else
    AGVCalc.Charge = [AGVData.Charge];
    AGVCalc.Position = [AGVData.Position];
    AGVCalc.Time = zeros(size(AGVCalc.Charge));
    AGVCalc.select = zeros(size(AGVCalc.Charge));
    selectedAGV = 1 ;
end

% %selecting proper AGV
%     AGVCalc.Charge = [AGVData.Charge,ChargeValue];
%     AGVCalc.Position = [AGVData.Position , 0];
%     AGVCalc.Time = zeros(size(AGVCalc.Charge));
%     AGVCalc.select = zeros(size(AGVCalc.Charge));
%     selectedAGV = 1 ;

% Checking the charge of each AGV and charging state of them
for j=1:size(AGVData.Charge,2)
    if AGVData.State(j) == 0 && AGVData.ReadyTime(j) <=
(CInput.ExeTime (i))
        AGVData.Charge(j) = ChargeValue;
        AGVData.Position(j) = 0;
        AGVData.State(j) = 1;
        AGVData.TimeOfCharging(j) = 0;
        AGVData.TimesCharged (j) = AGVData.TimesCharged (j) + 1;
    end
end
for j=1:size(AGVData.Charge,2)

```



```

        if AGVData.Charge(j) < ...

(TravTime(AGVData.Position(j)+1,CInput.StartPoint(i)+1) + ...
    TravTime(CInput.StartPoint(i)+1,CInput.MN(i)+1))+ ...
    TravTime(CInput.MN(i)+1,1) * ...
    Gamma ...
    && AGVData.State(j) ~= 0
    AGVData.Charge(j) = AGVData.Charge(j) -
TravTime(AGVData.Position(j)+1,1)*Gamma;
    if AGVData.Charge(j) < 0
        if exist('Error', 'var')
            Error.AGVData = [Error.AGVData; AGVData];
            Error.AGVSelection = [Error.AGVSelection;
AGVCalc];

            Error.CInput = [Error.CInput; CInput];
        else
            Error.AGVData = AGVData;
            Error.AGVSelection = AGVCalc;
            Error.CInput = CInput;
        end
    end
    end
    AGVCalc.select(j) = 2;
    AGVData.State(j) = 0;
    AGVData.TimeOfCharging(j) = CInput.ExeTime(i) +
TravTime(AGVData.Position(j)+1,1);
    AGVData.ReadyTime(j) = AGVData.TimeOfCharging(j) +
ChargingTime;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j=1:size(AGVCalc.Charge,2)
    %testing the remainder of charge after operation
    AGVCalc.Time(j) =
TravTime(AGVCalc.Position(j)+1,CInput.StartPoint(i)+1) + ...
    TravTime(CInput.StartPoint(i)+1,CInput.MN(i)+1) * Gamma;

    AGVCalc.Charge(j) = AGVCalc.Charge(j) - StPToHomeCharge -
...
    AGVCalc.Time(j) * Gamma ;

    if j <= size(AGVData.State,2)
        if AGVData.State(j) == 0
            AGVCalc.select(j) = 2;
        end
    end
    %Selecting the AGV
    if (j > 1 && j < size(AGVCalc.Charge,2) && AGVCalc.select(j) ~=
2) && InitialAGV ~= 2
        if AGVCalc.Time(j) < AGVCalc.Time(selectedAGV) ||
AGVCalc.select(selectedAGV) == 2
            selectedAGV = j;
        end
    end
    if j == size(AGVCalc.Charge,2) && InitialAGV ~= 2
        if Alpha * AGVCalc.Time(j) < AGVCalc.Time(selectedAGV) ||
AGVCalc.select(selectedAGV) == 2
            selectedAGV = j;
            AGVData.Charge = [AGVData.Charge, ChargeValue];
            AGVData.Position = [AGVData.Position, 0];
            AGVData.State = [AGVData.State, 1];
            AGVData.TimeOfCharging = [AGVData.TimeOfCharging, 0];
            AGVData.ReadyTime = [AGVData.ReadyTime, 0];
            AGVData.TimesCharged = [AGVData.TimesCharged, 0];

```

```

        end
    end
    if (j > 1 && AGVCalc.select(j) ~= 2) && InitialAGV == 2
        if AGVCalc.Time(j) < AGVCalc.Time(selectedAGV) ||
AGVCalc.select(selectedAGV) == 2
            selectedAGV = j;
        end
    end
end
end
if min(AGVCalc.select) == 2
    [A, Index] = min(AGVData.ReadyTime);
    if CInput.ExeTime(i) < A
        CInput.ExeTime(i) = A;
        selectedAGV = Index;
        AGVData.Charge(selectedAGV) = ChargeValue;
        AGVData.Position(selectedAGV) = 0;
        AGVData.State(selectedAGV) = 1;
        AGVData.TimeOfCharging(selectedAGV) = 0;
        AGVData.TimesCharged(selectedAGV) = AGVData.TimesCharged
(selectedAGV) + 1;
    end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MachineDelayToUnload = 0;
for j=i-1:-1:1
    if CInput.MN(i) == CInput.MN(j)
        MachineDelayToUnload = CInput.TotalOpTime(j); %Previous
operation of the machine
        break;
    end
end
end

CInput.UnLAGVTime(i) = TravTime(AGVData.Position
(selectedAGV)+1,CInput.StartPoint(i)+1);

if CInput.ON(i) == 1
    CInput.SrAGVIdleTime(i) = 0;
    CInput.StAGVIdleTime(i) = MachineDelayToUnload -
(CInput.ExeTime(i) + ...
    CInput.SrAGVIdleTime(i) + CInput.UnLAGVTime(i) +
CInput.LAGVTime(i));

    if CInput.StAGVIdleTime(i) < 0
        CInput.StAGVIdleTime(i) = 0;
    end
else

    % calculate the starting Idle time
    CInput.SrAGVIdleTime(i) = CInput.TotalOpTime(POp) -
(CInput.ExeTime(i) + CInput.UnLAGVTime(i));
    if CInput.SrAGVIdleTime(i) < 0
        CInput.SrAGVIdleTime(i) = 0;
    end
    % calculate the Stopp ing Idle time
    CInput.StAGVIdleTime(i) = MachineDelayToUnload -
(CInput.ExeTime(i) + ...
    CInput.SrAGVIdleTime(i) + CInput.UnLAGVTime(i) +
CInput.LAGVTime(i));
    if CInput.StAGVIdleTime(i) < 0
        CInput.StAGVIdleTime(i) = 0;
    end
end
end
end

```

```

    CInput.TotalOpTime(i) = CInput.OT(i) + CInput.ExeTime(i) +
CInput.UnLAGVTime(i) + ...
        CInput.LAGVTime(i) + CInput.SrAGVIdleTime(i) +
CInput.StAGVIdleTime(i);

    AGVData.Charge(selectedAGV) = AGVData.Charge(selectedAGV) - ...
        Gamma * (CInput.UnLAGVTime(i) + CInput.LAGVTime(i));
    AGVData.Position(selectedAGV) = CInput.MN(i);
    CInput.AGVN(i) = selectedAGV;
    if i<size(CInput.JN,2)
        CInput.ExeTime(i+1) = CInput.ExeTime(i) + CInput.UnLAGVTime(i)
+ ...
        CInput.LAGVTime(i) + CInput.SrAGVIdleTime(i) +
CInput.StAGVIdleTime(i);
    end

    clear AGVCalc;
    DeadEnd = 0;

end
output.Y.Makespan(ieval,1) = max(CInput.TotalOpTime);
output.Y.NAGV(ieval,1) = size(AGVData.Charge,2);
AGVN = CInput.AGVN;
end

```

### Drawgraph

```
function DrawGraph( Record, finalReport, maxRun)
```

```

%% ----- Summary PSO -----
[finalReport.excel(1,1), IPSO] = min(finalReport.PSO.Eval(:,1));
[finalReport.excel(1,2), J] = max(finalReport.PSO.Eval(:,1));
finalReport.excel(1,3) = mean(finalReport.PSO.Eval(:,1));
finalReport.excel(1,4) = std(finalReport.PSO.Eval(:,1));
finalReport.excel(1,5) = IPSO;
finalReport.excel(1,6) = J;

finalReport.excel(2,1) = finalReport.PSO.Makespan(IPSO,1);
finalReport.excel(2,2) = finalReport.PSO.Makespan(J,1);
finalReport.excel(2,3) = mean(finalReport.PSO.Makespan(:,1));
finalReport.excel(2,4) = std(finalReport.PSO.Makespan(:,1));

finalReport.excel(3,1) = finalReport.PSO.NAGV(IPSO,1);
finalReport.excel(3,2) = finalReport.PSO.NAGV(J,1);
finalReport.excel(3,3) = mean(finalReport.PSO.NAGV(:,1));
finalReport.excel(3,4) = std(finalReport.PSO.NAGV(:,1));

%% ----- Summary GA -----
[finalReport.excel(4,1), IGA] = min(finalReport.GA.Eval(:,1));
[finalReport.excel(4,2), J] = max(finalReport.GA.Eval(:,1));
finalReport.excel(4,3) = mean(finalReport.GA.Eval(:,1));
finalReport.excel(4,4) = std(finalReport.GA.Eval(:,1));
finalReport.excel(4,5) = IGA;
finalReport.excel(4,6) = J;

finalReport.excel(5,1) = finalReport.GA.Makespan(IGA,1);
finalReport.excel(5,2) = finalReport.GA.Makespan(J,1);
finalReport.excel(5,3) = mean(finalReport.GA.Makespan(:,1));
finalReport.excel(5,4) = std(finalReport.GA.Makespan(:,1));

finalReport.excel(6,1) = finalReport.GA.NAGV(IGA,1);
finalReport.excel(6,2) = finalReport.GA.NAGV(J,1);
finalReport.excel(6,3) = mean(finalReport.GA.NAGV(:,1));

```

```

        finalReport.excel(6,4) = std(finalReport.GA.NAGV(:,1));

%% ----- Summary Hybrid GA PSO -----
    [finalReport.excel(7,1), IHGAPSO] =
    min(finalReport.HGAPSO.Eval(:,1));
    [finalReport.excel(7,2), J] = max(finalReport.HGAPSO.Eval(:,1));
    finalReport.excel(7,3) = mean(finalReport.HGAPSO.Eval(:,1));
    finalReport.excel(7,4) = std(finalReport.HGAPSO.Eval(:,1));
    finalReport.excel(7,5) = IHGAPSO;
    finalReport.excel(7,6) = J;

    finalReport.excel(8,1) = finalReport.HGAPSO.Makespan(IHGAPSO,1);
    finalReport.excel(8,2) = max(finalReport.HGAPSO.Makespan(J,1));
    finalReport.excel(8,3) = mean(finalReport.HGAPSO.Makespan(:,1));
    finalReport.excel(8,4) = std(finalReport.HGAPSO.Makespan(:,1));

    finalReport.excel(9,1) = finalReport.HGAPSO.NAGV(IHGAPSO,1);
    finalReport.excel(9,2) = finalReport.HGAPSO.NAGV(J,1);
    finalReport.excel(9,3) = mean(finalReport.HGAPSO.NAGV(:,1));
    finalReport.excel(9,4) = std(finalReport.HGAPSO.NAGV(:,1));

%% ----- Excel Report -----
    xlswrite('FinalReport.xlsx', finalReport.excel, 1, 'C2');

if maxRun > 1
    PSO = Record(IPSO).PSO;
    GA = Record(IGA).GA;
    HGAPSO = Record(IHGAPSO).HGAPSO;
else
    PSO = Record(1).PSO;
    GA = Record(1).GA;
    HGAPSO = Record(1).HGAPSO;
end

%% ----- Draw Graph -----
    CountPSO = [1:PSO.Parameter.MaxGen];
    CountGA = [1:GA.Parameter.MaxGen];
    CountHGAPSO = [1:HGAPSO.Parameter.MaxGen];

    MAXIMUM = max([max(PSO.output.History.Y.Eval),
max(GA.output.History.Y.Eval), max(HGAPSO.output.History.Y.Eval)]);
    DIGIT = ceil(log10(abs(MAXIMUM))/2);
    Max_Axe = ceil(MAXIMUM/(10^DIGIT))*10^DIGIT;
    MINIMUM = min([min(PSO.output.History.Y.Eval),
min(GA.output.History.Y.Eval), min(HGAPSO.output.History.Y.Eval)]);
    DIGIT = ceil(log10(abs(MINIMUM))/2);
    Min_Axe = -20+floor(MINIMUM/(10^DIGIT))*10^DIGIT;

%% ----- Plot GA -----
    GAPLOT = zeros(GA.Parameter.MaxGen,3);
    for i = 1 : GA.Parameter.MaxGen-1
        GAPLOT(i,1) = min(GA.output.History.Y.Eval((i-
1)*GA.Parameter.N+1) : (i*GA.Parameter.N));
        GAPLOT(i,2) = max(GA.output.History.Y.Eval((i-
1)*GA.Parameter.N+1) : (i*GA.Parameter.N));
        GAPLOT(i,3) = sum(GA.output.History.Y.Eval((i-
1)*GA.Parameter.N+1) : (i*GA.Parameter.N))/GA.Parameter.N;
    end
    GAPLOT(GA.Parameter.MaxGen,1) = min(
GA.output.History.Y.Eval((GA.Parameter.MaxGen-1)*GA.Parameter.N+1) :
(GA.Parameter.MaxGen*GA.Parameter.N));

```

```

        GAPLOT(GA.Parameter.MaxGen,2) = max(
GA.output.History.Y.Eval(((GA.Parameter.MaxGen-1)*GA.Parameter.N+1) :
(GA.Parameter.MaxGen*GA.Parameter.N));
        GAPLOT(GA.Parameter.MaxGen,3) = sum(
GA.output.History.Y.Eval(((GA.Parameter.MaxGen-1)*GA.Parameter.N+1) :
(GA.Parameter.MaxGen*GA.Parameter.N)))/GA.Parameter.N;

        figure('Name','GA','NumberTitle','off')
        plot ( CountGA, GAPLOT(:,1),'b',CountGA, GAPLOT(:,2),'r',CountGA,
GAPLOT(:,3),'c')
        axis([0,GA.Parameter.MaxGen-1,Min_Axe,Max_Axe])
        legend('Minimum','Maximum','Mean');

%       figure('Name','GA Minimum','NumberTitle','off')
%       plot ( CountGA, GAPLOT(:,1))
%       axis([0,GA.Parameter.MaxGen-1,Min_Axe,Max_Axe])
%       figure('Name','GA Maximum','NumberTitle','off')
%       plot ( CountGA, GAPLOT(:,2))
%       axis([0,GA.Parameter.MaxGen-1,Min_Axe,Max_Axe])
%       figure('Name','GA Mean','NumberTitle','off')
%       plot ( CountGA, GAPLOT(:,3))
%       axis([0,GA.Parameter.MaxGen-1,Min_Axe,Max_Axe])

%% ----- Plot PSO -----
        PSOPLOT = zeros(PSO.Parameter.MaxGen,3);
        for i = 1 : PSO.Parameter.MaxGen-1
            PSOPLOT(i,1) = min( PSO.output.History.Y.Eval(((i-
1)*PSO.Parameter.N+1) : (i*PSO.Parameter.N));
            PSOPLOT(i,2) = max( PSO.output.History.Y.Eval(((i-
1)*PSO.Parameter.N+1) : (i*PSO.Parameter.N));
            PSOPLOT(i,3) = sum( PSO.output.History.Y.Eval(((i-
1)*PSO.Parameter.N+1) : (i*PSO.Parameter.N)))/PSO.Parameter.N;
        end
        PSOPLOT(PSO.Parameter.MaxGen,1) = min(
PSO.output.History.Y.Eval(((PSO.Parameter.MaxGen-1)*PSO.Parameter.N+1)
: (PSO.Parameter.MaxGen*PSO.Parameter.N));
        PSOPLOT(PSO.Parameter.MaxGen,2) = max(
PSO.output.History.Y.Eval(((PSO.Parameter.MaxGen-1)*PSO.Parameter.N+1)
: (PSO.Parameter.MaxGen*PSO.Parameter.N));
        PSOPLOT(PSO.Parameter.MaxGen,3) = sum(
PSO.output.History.Y.Eval(((PSO.Parameter.MaxGen-1)*PSO.Parameter.N+1)
: (PSO.Parameter.MaxGen*PSO.Parameter.N)))/PSO.Parameter.N;

        figure('Name','PSO','NumberTitle','off')
        plot ( CountPSO, PSOPLOT(:,1),'b',CountPSO,
PSOPLOT(:,2),'r',CountPSO, PSOPLOT(:,3),'c')
        axis([0,PSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])
        legend('Minimum','Maximum','Mean');

%       figure('Name','PSO Minimum','NumberTitle','off')
%       plot ( CountPSO, PSOPLOT(:,1))
%       axis([0,PSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])
%       figure('Name','PSO Maximum','NumberTitle','off')
%       plot ( CountPSO, PSOPLOT(:,2))
%       axis([0,PSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])
%       figure('Name','PSO Mean','NumberTitle','off')
%       plot ( CountPSO, PSOPLOT(:,3))
%       axis([0,PSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])

%% ----- Plot Hybrid GA-PSO1 -----
        HGP2PLOT = zeros(HGP2.Parameter.MaxGen,3);
        for i = 1 : HGAPSO.Parameter.MaxGen-1

```

```

HGAPSO_PLOT(i,1) = min( HGAPSO.output.History.Y.Eval(((i-
1)*HGAPSO.Parameter.N+1) : (i*HGAPSO.Parameter.N)));
HGAPSO_PLOT(i,2) = max( HGAPSO.output.History.Y.Eval(((i-
1)*HGAPSO.Parameter.N+1) : (i*HGAPSO.Parameter.N)));
HGAPSO_PLOT(i,3) = sum( HGAPSO.output.History.Y.Eval(((i-
1)*HGAPSO.Parameter.N+1) :
(i*HGAPSO.Parameter.N)))/HGAPSO.Parameter.N;
end

HGAPSO_PLOT(HGAPSO.Parameter.MaxGen,1) = min(
HGAPSO.output.History.Y.Eval(((HGAPSO.Parameter.MaxGen-
1)*HGAPSO.Parameter.N+1) :
(HGAPSO.Parameter.MaxGen*HGAPSO.Parameter.N)));
HGAPSO_PLOT(HGAPSO.Parameter.MaxGen,2) = max(
HGAPSO.output.History.Y.Eval(((HGAPSO.Parameter.MaxGen-
1)*HGAPSO.Parameter.N+1) :
(HGAPSO.Parameter.MaxGen*HGAPSO.Parameter.N)));
HGAPSO_PLOT(HGAPSO.Parameter.MaxGen,3) = sum(
HGAPSO.output.History.Y.Eval(((HGAPSO.Parameter.MaxGen-
1)*HGAPSO.Parameter.N+1) :
(HGAPSO.Parameter.MaxGen*HGAPSO.Parameter.N)))/HGAPSO.Parameter.N;

figure('Name','Hybrid GA-PSO','NumberTitle','off')
plot ( CountHGAPSO, HGAPSO_PLOT(:,1),'b',CountHGAPSO,
HGAPSO_PLOT(:,2),'r',CountHGAPSO, HGAPSO_PLOT(:,3),'c')
axis([0,HGAPSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])
legend('Minimum','Maximum','Mean');

% figure('Name','Hybrid GA-PSO Minimum','NumberTitle','off')
% plot ( CountHGAPSO, HGAPSO_PLOT(:,1))
% axis([0,HGAPSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])
% figure('Name','Hybrid GA-PSO Maximum','NumberTitle','off')
% plot ( CountHGAPSO, HGAPSO_PLOT(:,2))
% axis([0,HGAPSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])
% figure('Name','Hybrid GA-PSO Mean','NumberTitle','off')
% plot ( CountHGAPSO, HGAPSO_PLOT(:,3))
% axis([0,HGAPSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])

%% ----- Plot Hybrid GA-PSO2 -----
HGP2_PLOT = zeros(HGP2.Parameter.MaxGen,3);
for i = 1 : HGAPSO.Parameter.MaxGen-1
HGAPSO_PLOT(i,1) = min( HGAPSO.output.History.Y.Eval(((i-
1)*HGAPSO.Parameter.N+1) : (i*HGAPSO.Parameter.N)));
HGAPSO_PLOT(i,2) = max( HGAPSO.output.History.Y.Eval(((i-
1)*HGAPSO.Parameter.N+1) : (i*HGAPSO.Parameter.N)));
HGAPSO_PLOT(i,3) = sum( HGAPSO.output.History.Y.Eval(((i-
1)*HGAPSO.Parameter.N+1) :
(i*HGAPSO.Parameter.N)))/HGAPSO.Parameter.N;
end

HGAPSO_PLOT(HGAPSO.Parameter.MaxGen,1) = min(
HGAPSO.output.History.Y.Eval(((HGAPSO.Parameter.MaxGen-
1)*HGAPSO.Parameter.N+1) :
(HGAPSO.Parameter.MaxGen*HGAPSO.Parameter.N)));
HGAPSO_PLOT(HGAPSO.Parameter.MaxGen,2) = max(
HGAPSO.output.History.Y.Eval(((HGAPSO.Parameter.MaxGen-
1)*HGAPSO.Parameter.N+1) :
(HGAPSO.Parameter.MaxGen*HGAPSO.Parameter.N)));
HGAPSO_PLOT(HGAPSO.Parameter.MaxGen,3) = sum(
HGAPSO.output.History.Y.Eval(((HGAPSO.Parameter.MaxGen-
1)*HGAPSO.Parameter.N+1) :
(HGAPSO.Parameter.MaxGen*HGAPSO.Parameter.N)))/HGAPSO.Parameter.N;

```

```

    figure('Name','Hybrid GA-PSO','NumberTitle','off')
    plot ( CountHGAPSO, HGAPSO_PLOT(:,1),'b',CountHGAPSO,
HGAPSO_PLOT(:,2),'r',CountHGAPSO, HGAPSO_PLOT(:,3),'c')
    axis([0,HGAPSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])
    legend('Minimum','Maximum','Mean');

%     figure('Name','Hybrid GA-PSO Minimum','NumberTitle','off')
%     plot ( CountHGAPSO, HGAPSO_PLOT(:,1))
%     axis([0,HGAPSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])
%     figure('Name','Hybrid GA-PSO Maximum','NumberTitle','off')
%     plot ( CountHGAPSO, HGAPSO_PLOT(:,2))
%     axis([0,HGAPSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])
%     figure('Name','Hybrid GA-PSO Mean','NumberTitle','off')
%     plot ( CountHGAPSO, HGAPSO_PLOT(:,3))
%     axis([0,HGAPSO.Parameter.MaxGen-1,Min_Axe,Max_Axe])

%% ----- Plot All -----
    figure('Name','Minimum','NumberTitle','off')
    plot ( CountHGP2, HGP2_PLOT(:,1),'b')
    hold on;
    plot ( CountHGP1, HGP1_PLOT(:,1),'d')
    hold on;
    plot ( CountPSO, PSO_PLOT(:,1),'r')
    hold on;
    plot ( CountGA, GA_PLOT(:,1), 'c')
    legend('HGAPSO','PSO','GA');
end

```

### Mat2gray

```

function I = mat2gray(A,limits)
%MAT2GRAY Convert matrix to intensity image.
% I = MAT2GRAY(A,[AMIN AMAX]) converts the matrix A to the intensity
image I.
% The returned matrix I contains values in the range 0.0 (black) to
1.0 (full
% intensity or white). AMIN and AMAX are the values in A that
correspond to
% 0.0 and 1.0 in I. Values less than AMIN become 0.0, and values
greater than
% AMAX become 1.0.
%
% I = MAT2GRAY(A) sets the values of AMIN and AMAX to the minimum
and maximum
% values in A.
%
% Class Support
% -----
% The input array A can be logical or numeric. The output image I is
double.
%
% Example
% -----
%     I = imread('rice.png');
%     J = filter2(fspecial('sobel'), I);
%     K = mat2gray(J);
%     figure, imshow(I), figure, imshow(K)

% Copyright 1992-2014 The MathWorks, Inc.

validateattributes(A,{'logical','uint8','uint16','uint32',...
'int8','int16','int32','single','double'},...
{'},mfilename,'A',1);

```

```

if nargin == 1
    limits = double([min(A(:)) max(A(:))]);
else
    validateattributes(limits,{'double'},{'numel',2},mfilename,'LIMITS',2)
;
end

if limits(2)==limits(1)    % Constant Image
    I = double(A);
else
    delta = 1 / (limits(2) - limits(1));
    I = imlincomb(delta, A, -limits(1)*delta, 'double');
end

% Make sure all values in I are between 0 and 1.
I = max(0,min(I,1));

```

### **Imlincomb**

```

function Z = imlincomb(varargin)
%IMLINCOMB Linear combination of images.
% Z = IMLINCOMB(K1,A1,K2,A2, ..., Kn,An) computes K1*A1 + K2*A2 +
... +
% Kn*An. A1, A2, ..., An are real, non-sparse, numeric arrays with
the
% same class and size, and K1, K2, ..., Kn are real double scalars.
Z
% has the same size and class as A1 unless A1 is logical, in which
case
% Z is double.
%
% Z = IMLINCOMB(K1,A1,K2,A2, ..., Kn,An,K) computes K1*A1 + K2*A2 +
... + Kn*An + K.
%
% Z = IMLINCOMB(..., OUTPUT_CLASS) lets you specify the class of Z.
% OUTPUT_CLASS is a string containing the name of a numeric class.
%
% Each element of the output, Z, is computed individually in
% double-precision floating point. When Z is an integer array,
elements
% of Z that exceed the range of the integer type are truncated, and
% fractional values are rounded.
%
% Example 1
% -----
% Scale an image by a factor of two.
%
%     I = imread('cameraman.tif');
%     J = imlincomb(2,I);
%     figure, imshow(J)
%
% Example 2
% -----
% Form a difference image with the zero value shifted to 128.
%
%     I = imread('cameraman.tif');
%     J = uint8(filter2(fspecial('gaussian'), I));
%     K = imlincomb(1,I,-1,J,128); % K(r,c) = I(r,c) - J(r,c) + 128
%     figure, imshow(K)
%
% Example 3
% -----
% Add two images with a specified output class.

```



```

%
%     I = imread('rice.png');
%     J = imread('cameraman.tif');
%     K = imlincomb(1,I,1,J,'uint16');
%     figure, imshow(K,[])
%
% See also IMCOMPLEMENT.

% Copyright 1993-2014 The MathWorks, Inc.

% I/O spec
% =====
% A1, ...      Real, numeric, full arrays
%              Logical arrays also allowed, and are converted to
uint8.
%
% K1, ...      Real, double scalars
%
% OUTPUT_CLASS Case-insensitive nonambiguous abbreviation of one of
%              these strings: uint8, uint16, uint32, int8, int16,
int32,
%              single, double

[ims, scalars, outputClass] = ParseInputs(varargin{:});

sameInputOutputClass = strcmp(class(ims{1}), outputClass);
if sameInputOutputClass
    if imagePlusImage(ims, scalars)
        Z = ims{1} + ims{2};
    elseif image1MinusImage2(ims, scalars)
        Z = ims{1} - ims{2};
    elseif image2MinusImage1(ims, scalars)
        Z = ims{2} - ims{1};
    elseif imagePlusScalar(ims, scalars)
        Z = ims{1} + scalars(2);
    else
        Z = images.internal.imlincombc(ims, scalars, outputClass);
    end
else
    Z = images.internal.imlincombc(ims, scalars, outputClass);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function valid = imagePlusImage(images, scalars)

    valid = numel(images) == 2 && numel(scalars) == 2 && ...
        all(scalars == 1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function valid = image1MinusImage2(images, scalars)

    valid = numel(images) == 2 && numel(scalars) == 2 && ...
        scalars(1) == 1 && scalars(2) == -1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function valid = image2MinusImage1(images, scalars)

    valid = numel(images) == 2 && numel(scalars) == 2 && ...
        scalars(1) == -1 && scalars(2) == 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function valid = imagePlusScalar(images, scalars)

```

```

        valid = numel(images) == 1 && numel(scalars) == 2 && ...
            scalars(1) == 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [images, scalars, output_class] = ParseInputs(varargin)

marginchk(2, Inf);

if ischar(varargin{end})
    valid_strings = {'uint8' 'uint16' 'uint32' 'int8' 'int16' 'int32'
        ...
            'single' 'double'};
    output_class = validatestring(varargin{end}, valid_strings,
mfilename, ...
            'OUTPUT_CLASS', 3);
    varargin(end) = [];
else
    if islogical(varargin{2})
        output_class = 'double';
    else
        output_class = class(varargin{2});
    end
end

%check images
images = varargin(2:2:end);
if ~iscell(images) || isempty(images)
    displayInternalError('images');
end

% assign and check scalars
for p = 1:2:length(varargin)
    validateattributes(varargin{p}, {'double'}, {'real' 'nonsparse'
'scalar'}, ...
            mfilename, sprintf('K%d', (p+1)/2), p);
end
scalars = [varargin{1:2:end}];

%make sure it is a vector
if ( ~ismatrix(scalars) || (all(size(scalars)~=1) &&
any(size(scalars)~=0)) )
    displayInternalError('scalars');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function displayInternalError(string)

error(message('images:imlincomb:internalError', upper( string )))

```

### Reportrecord

```

function Record = ReportRecord( Record )
MaxRun = length(Record);
savedata = isfield(Record, 'finalReport');

for run = 1:MaxRun
    PSO = Record(run).PSO;
    GA = Record(run).GA;
    HGAPSO = Record(run).HGAPSO;
    [PSO.output.Best.Y.Eval, I] = min(PSO.output.History.Y.Eval);

```

```

    PSO.output.Best.Y.Makespan = PSO.output.History.Y.Makespan(I);
    PSO.output.Best.Y.NAGV = PSO.output.History.Y.NAGV(I);
    PSO.output.Best.X = PSO.output.History.X(I+1,:);
    fprintf('The PSO results with "%6.0f " function evaluation
\n', PSO.Parameter.MaxGen * PSO.Parameter.N );
    fprintf('Makespan =      %9.0f \nNumber of AGV =      %3.0f \n',
PSO.output.Best.Y.Makespan, PSO.output.Best.Y.NAGV);
    fprintf('time =                %4.4f Sec\nBest Generation in %5.0f
\n', PSO.time, I);
    fprintf('\n-----
\n\n');

    PSO.output.Worst.Y.Eval = max(PSO.output.History.Y.Eval);
    PSO.output.Worst.Y.Makespan = max(PSO.output.History.Y.Makespan);
    PSO.output.Worst.Y.NAGV = max(PSO.output.History.Y.NAGV);

    PSO.output.Mean.Y.Eval = mean(PSO.output.History.Y.Eval);
    PSO.output.Mean.Y.Makespan = mean(PSO.output.History.Y.Makespan);
    PSO.output.Mean.Y.NAGV = mean(PSO.output.History.Y.NAGV);

    PSO.output.ST.Y.Eval = std(PSO.output.History.Y.Eval);
    PSO.output.ST.Y.Makespan = std(PSO.output.History.Y.Makespan);
    PSO.output.ST.Y.NAGV = std(PSO.output.History.Y.NAGV);

    finalReport.PSO.Eval(run,:) = [PSO.output.Best.Y.Eval,
PSO.output.Worst.Y.Eval, PSO.output.Mean.Y.Eval,
PSO.output.ST.Y.Eval];
    finalReport.PSO.Makespan(run,:) = [PSO.output.Best.Y.Makespan,
PSO.output.Worst.Y.Makespan, PSO.output.Mean.Y.Makespan,
PSO.output.ST.Y.Makespan];
    finalReport.PSO.NAGV(run,:) = [PSO.output.Best.Y.NAGV,
PSO.output.Worst.Y.NAGV, PSO.output.Mean.Y.NAGV,
PSO.output.ST.Y.NAGV];

    Best.PSO.X(run,:) = PSO.output.Best.X;
    Best.PSO.Eval(run,:) = PSO.output.Best.Y.Eval;
    Best.PSO.Makespan(run,:) = PSO.output.Best.Y.Makespan;
    Best.PSO.NAGV(run,:) = PSO.output.Best.Y.NAGV;

    Record(run).PSO = PSO;

%% -----Report GA-----
    [GA.output.Best.Y.Eval, I] = min(GA.output.History.Y.Eval);
    GA.output.Best.Y.Makespan = GA.output.History.Y.Makespan(I);
    GA.output.Best.Y.NAGV = GA.output.History.Y.NAGV(I);
    GA.output.Best.X = GA.output.History.X(I+1,:);
    fprintf('The GA results with "%6.0f " function evaluation
\n', GA.Parameter.MaxGen * GA.Parameter.N );
    fprintf('Makespan =      %9.0f \nNumber of AGV =      %3.0f \n',
GA.output.Best.Y.Makespan, GA.output.Best.Y.NAGV);
    fprintf('time =                %4.4f Sec\nBest Generation in %5.0f
\n', GA.time, I);
    fprintf('\n-----
\n\n');

    GA.output.Worst.Y.Eval = max(GA.output.History.Y.Eval);
    GA.output.Worst.Y.Makespan = max(GA.output.History.Y.Makespan);
    GA.output.Worst.Y.NAGV = max(GA.output.History.Y.NAGV);

    GA.output.Mean.Y.Eval = mean(GA.output.History.Y.Eval);
    GA.output.Mean.Y.Makespan = mean(GA.output.History.Y.Makespan);
    GA.output.Mean.Y.NAGV = mean(GA.output.History.Y.NAGV);

```

```

GA.output.ST.Y.Eval = std(GA.output.History.Y.Eval);
GA.output.ST.Y.Makespan = std(GA.output.History.Y.Makespan);
GA.output.ST.Y.NAGV = std(GA.output.History.Y.NAGV);

finalReport.GA.Eval(run,:) = [GA.output.Best.Y.Eval,
GA.output.Worst.Y.Eval, GA.output.Mean.Y.Eval, GA.output.ST.Y.Eval];
finalReport.GA.Makespan(run,:) = [GA.output.Best.Y.Makespan,
GA.output.Worst.Y.Makespan, GA.output.Mean.Y.Makespan,
GA.output.ST.Y.Makespan];
finalReport.GA.NAGV(run,:) = [GA.output.Best.Y.NAGV,
GA.output.Worst.Y.NAGV, GA.output.Mean.Y.NAGV, GA.output.ST.Y.NAGV];

Best.GA.X(run,:) = GA.output.Best.X;
Best.GA.Eval(run,:) = GA.output.Best.Y.Eval;
Best.GA.Makespan(run,:) = GA.output.Best.Y.Makespan;
Best.GA.NAGV(run,:) = GA.output.Best.Y.NAGV;

Record(run).GA = GA;
%% -----Report Hybrid GA - PSO -----

[HGAPSO.output.Best.Y.Eval, I] =
min(HGAPSO.output.History.Y.Eval);
HGAPSO.output.Best.Y.Makespan =
HGAPSO.output.History.Y.Makespan(I);
HGAPSO.output.Best.Y.NAGV = HGAPSO.output.History.Y.NAGV(I);
HGAPSO.output.Best.X = HGAPSO.output.History.X(I+1,:);
fprintf('The Hybrid GA-PSO results with "%6.0f " function
evaluation \n', HGAPSO.Parameter.MaxGen * HGAPSO.Parameter.N );
fprintf('Makespan = %9.0f \nNumber of AGV = %3.0f \n',
HGAPSO.output.Best.Y.Makespan, HGAPSO.output.Best.Y.NAGV);
fprintf('time = %4.4f Sec\nBest Generation in %5.0f
\n', HGAPSO.time,I);
fprintf('\n-----
\n\n');

HGAPSO.output.Worst.Y.Eval = max(HGAPSO.output.History.Y.Eval);
HGAPSO.output.Worst.Y.Makespan =
max(HGAPSO.output.History.Y.Makespan);
HGAPSO.output.Worst.Y.NAGV = max(HGAPSO.output.History.Y.NAGV);

HGAPSO.output.Mean.Y.Eval = mean(HGAPSO.output.History.Y.Eval);
HGAPSO.output.Mean.Y.Makespan =
mean(HGAPSO.output.History.Y.Makespan);
HGAPSO.output.Mean.Y.NAGV = mean(HGAPSO.output.History.Y.NAGV);

HGAPSO.output.ST.Y.Eval = std(HGAPSO.output.History.Y.Eval);
HGAPSO.output.ST.Y.Makespan =
std(HGAPSO.output.History.Y.Makespan);
HGAPSO.output.ST.Y.NAGV = std(HGAPSO.output.History.Y.NAGV);

finalReport.HGAPSO.Eval(run,:) = [HGAPSO.output.Best.Y.Eval,
HGAPSO.output.Worst.Y.Eval, HGAPSO.output.Mean.Y.Eval,
HGAPSO.output.ST.Y.Eval];
finalReport.HGAPSO.Makespan(run,:) =
[HGAPSO.output.Best.Y.Makespan, HGAPSO.output.Worst.Y.Makespan,
HGAPSO.output.Mean.Y.Makespan, HGAPSO.output.ST.Y.Makespan];
finalReport.HGAPSO.NAGV(run,:) = [HGAPSO.output.Best.Y.NAGV,
HGAPSO.output.Worst.Y.NAGV, HGAPSO.output.Mean.Y.NAGV,
HGAPSO.output.ST.Y.NAGV];

Best.HGAPSO.X(run,:) = HGAPSO.output.Best.X;

```

```

Best.HGAPSO.Eval(run,:) = HGAPSO.output.Best.Y.Eval;
Best.HGAPSO.Makespan(run,:) = HGAPSO.output.Best.Y.Makespan;
Best.HGAPSO.NAGV(run,:) = HGAPSO.output.Best.Y.NAGV;

Record(run).HGAPSO = HGAPSO;

Record(run).finalReport = finalReport;
Record(run).Best = Best;
end

[B, I] = min(Record(end).Best.PSO.Eval);
Record(end).BestOfBest.PSO.Eval = B;
Record(end).BestOfBest.PSO.X = Record(end).Best.PSO.X(I,:);
Record(end).BestOfBest.PSO.Makespan =
Record(end).Best.PSO.Makespan(I);
Record(end).BestOfBest.PSO.NAGV = Record(end).Best.PSO.NAGV(I);

[B, I] = min(Record(end).Best.GA.Eval);
Record(end).BestOfBest.GA.Eval = B;
Record(end).BestOfBest.GA.X = Record(end).Best.GA.X(I,:);
Record(end).BestOfBest.GA.Makespan = Record(end).Best.GA.Makespan(I);
Record(end).BestOfBest.GA.NAGV = Record(end).Best.GA.NAGV(I);

[B, I] = min(Record(end).Best.HGAPSO.Eval);
Record(end).BestOfBest.HGAPSO.Eval = B;
Record(end).BestOfBest.HGAPSO.X = Record(end).Best.HGAPSO.X(I,:);
Record(end).BestOfBest.HGAPSO.Makespan =
Record(end).Best.HGAPSO.Makespan(I);
Record(end).BestOfBest.HGAPSO.NAGV = Record(end).Best.HGAPSO.NAGV(I);

DrawGraph( Record, finalReport, MaxRun);

%% ----- Save Data -----
if savedata == false
    filename = strcat('Record_', num2str(sum(clock)*1000), '.mat');
    save(filename,'Record');
end

```

### Reportit

```

function [out]= Reportit(this)

    if this.output.History.Y.Makespan(1,1) == inf
        this.output.History.Y.Makespan(1,:) = [];
    end
    if this.output.History.Y.Eval(1,1) == inf
        this.output.History.Y.Eval(1,:) = [];
    end
    if this.output.History.Y.NAGV(1,1) == inf
        this.output.History.Y.NAGV(1,:) = [];
    end

    [this.output.Best.Y.Eval, I] = min(this.output.History.Y.Eval);
    this.output.Best.Y.Makespan = this.output.History.Y.Makespan(I);
    this.output.Best.Y.NAGV = this.output.History.Y.NAGV(I);
    this.output.Best.X = this.output.History.X(I,:);
    fprintf('The results with "%6.0f " function evaluation      \n',
this.Parameter.MaxGen * this.Parameter.N );
    fprintf('Makespan = %9.0f \nNumber of AGV = %3.0f \n',
this.output.Best.Y.Makespan, this.output.Best.Y.NAGV);
    fprintf('\n-----
\n\n');

```

```

%      this.output.Best.Y.Eval = min(this.output.History.Y.Eval);
%      this.output.Best.Y.Makespan =
min(this.output.History.Y.Makespan);
%      this.output.Best.Y.NAGV = min(this.output.History.Y.NAGV);

      this.output.Worst.Y.Eval = max(this.output.History.Y.Eval);
      this.output.Worst.Y.Makespan =
max(this.output.History.Y.Makespan);
      this.output.Worst.Y.NAGV = max(this.output.History.Y.NAGV);

      this.output.Mean.Y.Eval = mean(this.output.History.Y.Eval);
      this.output.Mean.Y.Makespan =
mean(this.output.History.Y.Makespan);
      this.output.Mean.Y.NAGV = mean(this.output.History.Y.NAGV);

      this.output.ST.Y.Eval = std(this.output.History.Y.Eval);
      this.output.ST.Y.Makespan = std(this.output.History.Y.Makespan);
      this.output.ST.Y.NAGV = std(this.output.History.Y.NAGV);

      Countthis = [1:this.Parameter.MaxGen];

      MAXIMUM = max(this.output.History.Y.Makespan);
      DIGIT = ceil(log10(abs(MAXIMUM))/2);
      Max_Axe = ceil(MAXIMUM/(10^DIGIT))*10^DIGIT;
      MINIMUM = min(this.output.History.Y.Makespan);
      DIGIT = ceil(log10(abs(MINIMUM))/2);
      Min_Axe = floor(MINIMUM/(10^DIGIT))*10^DIGIT;

%% ----- Plot this -----
      thisPLOT = zeros(this.Parameter.MaxGen,3);
      for i = 1 : this.Parameter.MaxGen-1
          thisPLOT(i,1) = min( this.output.History.Y.Makespan(((i-
1)*this.Parameter.N+1) : (i*this.Parameter.N)));
          thisPLOT(i,2) = max( this.output.History.Y.Makespan(((i-
1)*this.Parameter.N+1) : (i*this.Parameter.N)));
          thisPLOT(i,3) = sum( this.output.History.Y.Makespan(((i-
1)*this.Parameter.N+1) : (i*this.Parameter.N)))/this.Parameter.N;
      end
      thisPLOT(this.Parameter.MaxGen,1) = min(
this.output.History.Y.Makespan(((this.Parameter.MaxGen-
1)*this.Parameter.N+1) : (this.Parameter.MaxGen*this.Parameter.N)));
      thisPLOT(this.Parameter.MaxGen,2) = max(
this.output.History.Y.Makespan(((this.Parameter.MaxGen-
1)*this.Parameter.N+1) : (this.Parameter.MaxGen*this.Parameter.N)));
      thisPLOT(this.Parameter.MaxGen,3) = sum(
this.output.History.Y.Makespan(((this.Parameter.MaxGen-
1)*this.Parameter.N+1) :
(this.Parameter.MaxGen*this.Parameter.N)))/this.Parameter.N;

      figure('Name','this Minimum','NumberTitle','off')
      plot ( Countthis, thisPLOT(:,1))
      axis([0,this.Parameter.MaxGen-1,Min_Axe,Max_Axe])
      figure('Name','this Maximum','NumberTitle','off')
      plot ( Countthis, thisPLOT(:,2))
      axis([0,this.Parameter.MaxGen-1,Min_Axe,Max_Axe])
      figure('Name','this Mean','NumberTitle','off')
      plot ( Countthis, thisPLOT(:,3))
      axis([0,this.Parameter.MaxGen-1,Min_Axe,Max_Axe])

      figure('Name','All','NumberTitle','off')
      plot ( Countthis, thisPLOT(:,1))
      hold on;

```

```
plot ( Countthis, thisPLOT(:,2))  
hold on;  
plot ( Countthis, thisPLOT(:,3))  
out = 1;  
legend('min','max', 'mean');
```